From Object Replication to Database Replication

Fernando Pedone
University of Lugano
Switzerland

Outline

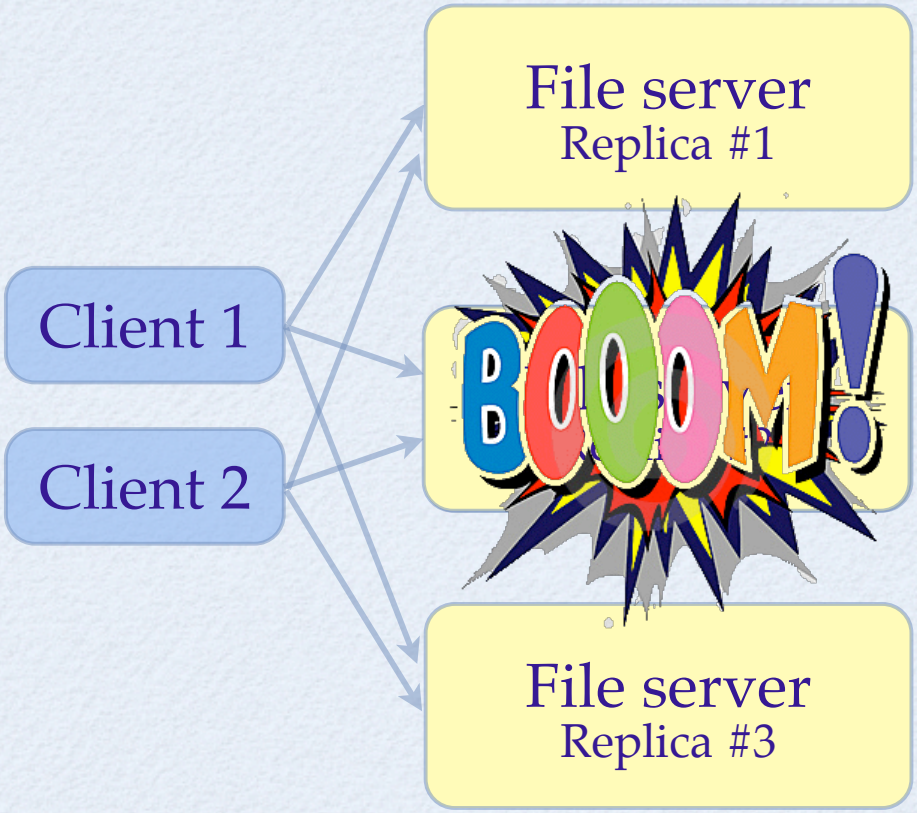
- Motivation
- Replication model
- From objects to databases
- Deferred update replication
- Final remarks

Motivation

- **From object replication...**
  - Replication in the distributed systems community
  - Processes, non-transactional objects
- **...to database replication**
  - Replication in the database community
  - Transactions, databases

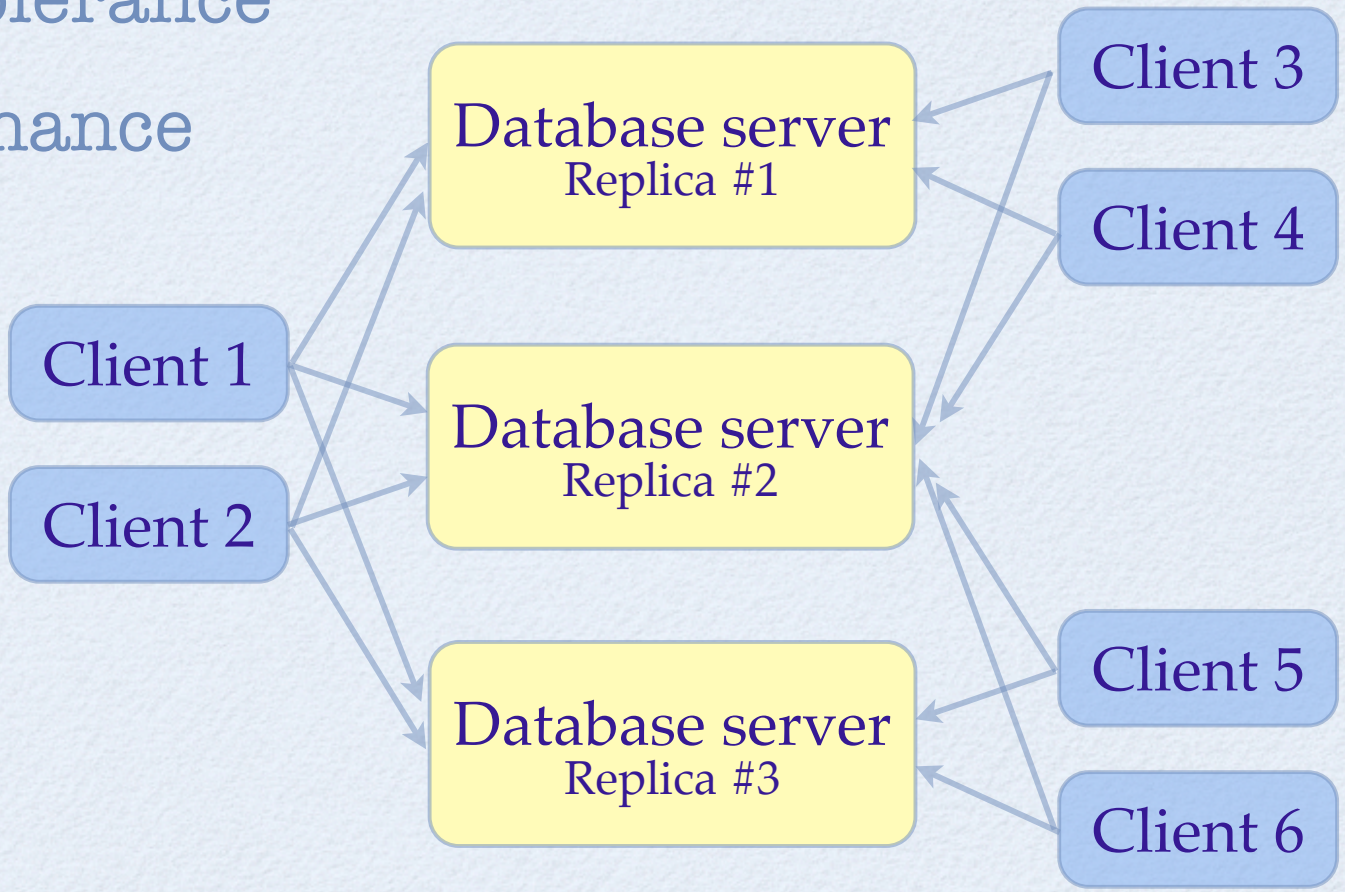
Motivation

- Object replication is important
  - Fault tolerance



Motivation

- Database replication is important
  - Fault tolerance
  - Performance

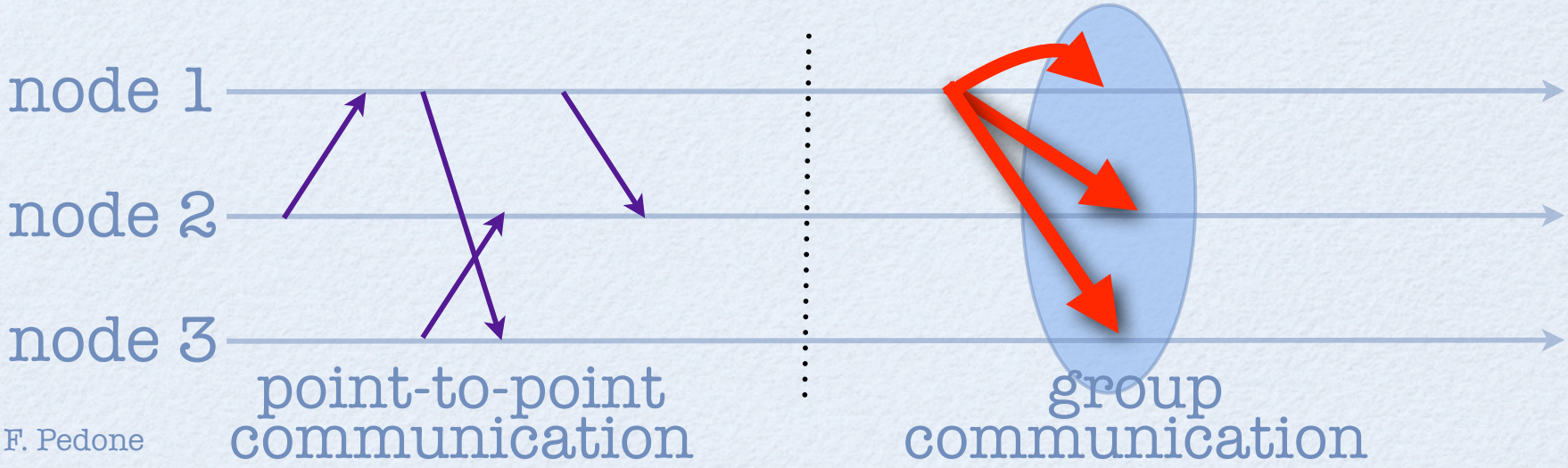


Motivation

- Object vs. database replication (recap)
  - Different goals
    - Fault tolerance vs. Performance & Fault tolerance
  - Different models
    - Objects, non-transactional vs. transactions
  - Different algorithms
    - to a certain extent...

Motivation

- Group communication
  - Messages addressed to a group of nodes
  - Initially used for object replication
  - Later used for database replication



Outline

- Motivation
- **Replication model**
- From objects to databases
- Deferred update replication
- Final remarks

Replication model

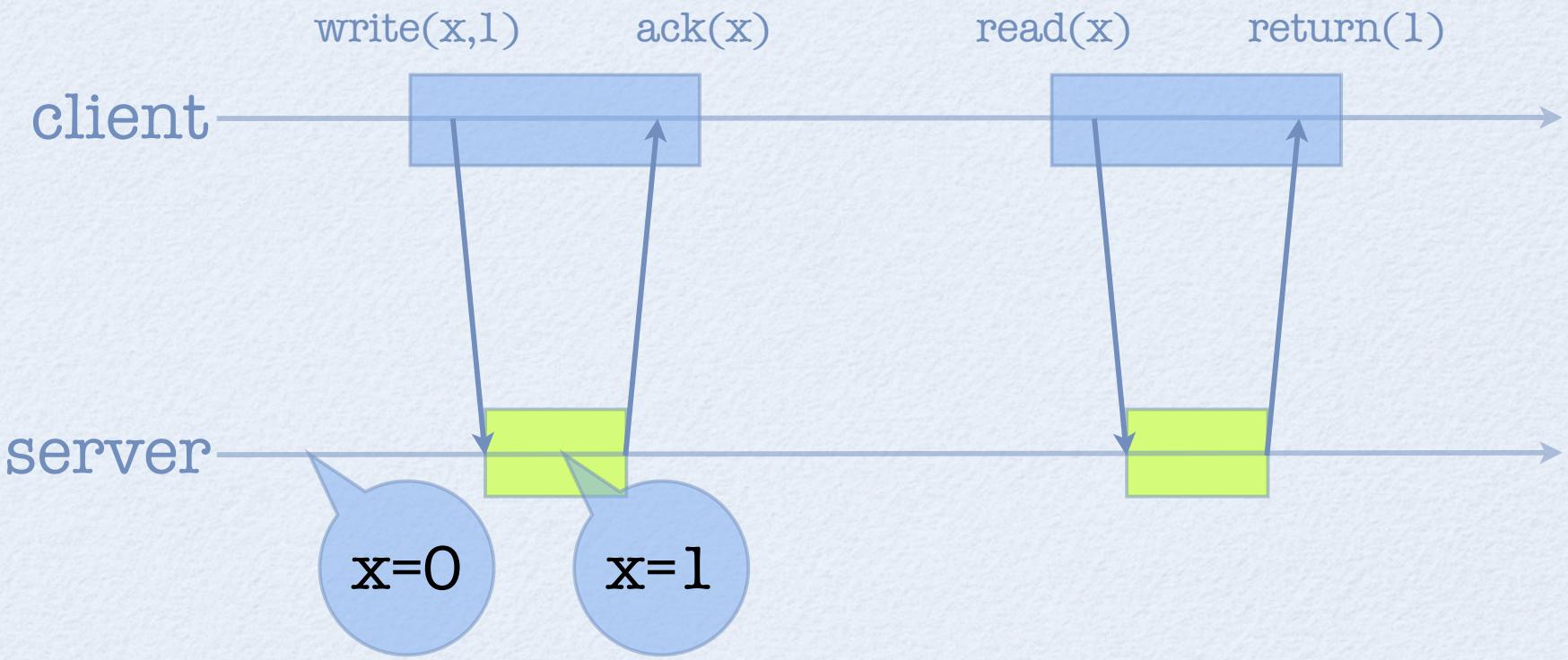
- **Object model**
  - Clients: c_1, c_2,...
  - Servers: s_1, ..., s_n
  - Operations: **read** and **write**
    - read(x) returns the value of object x
    - write(x,v) updates the value of x with v; returns acknowledgement

Replication model

- Object consistency
  - Consistency criteria
    - Defines the behavior of object operations
    - Simple for single-client-single-server case
    - But more complex in the general case
      - Multiple clients
      - Multiple servers
      - Failures

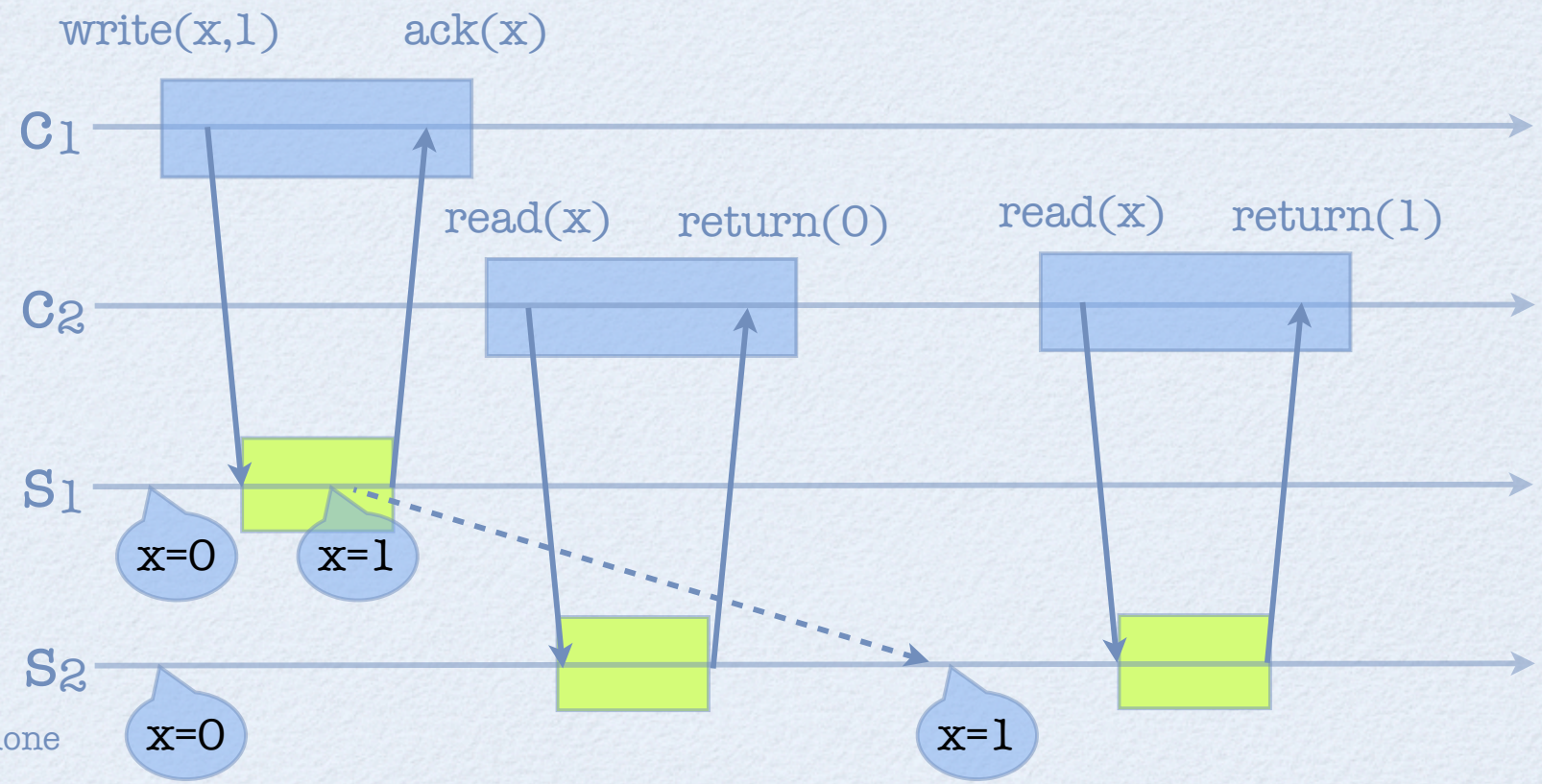
Replication model

- Consistency criteria
  - Simple case: single client, single server, no failures



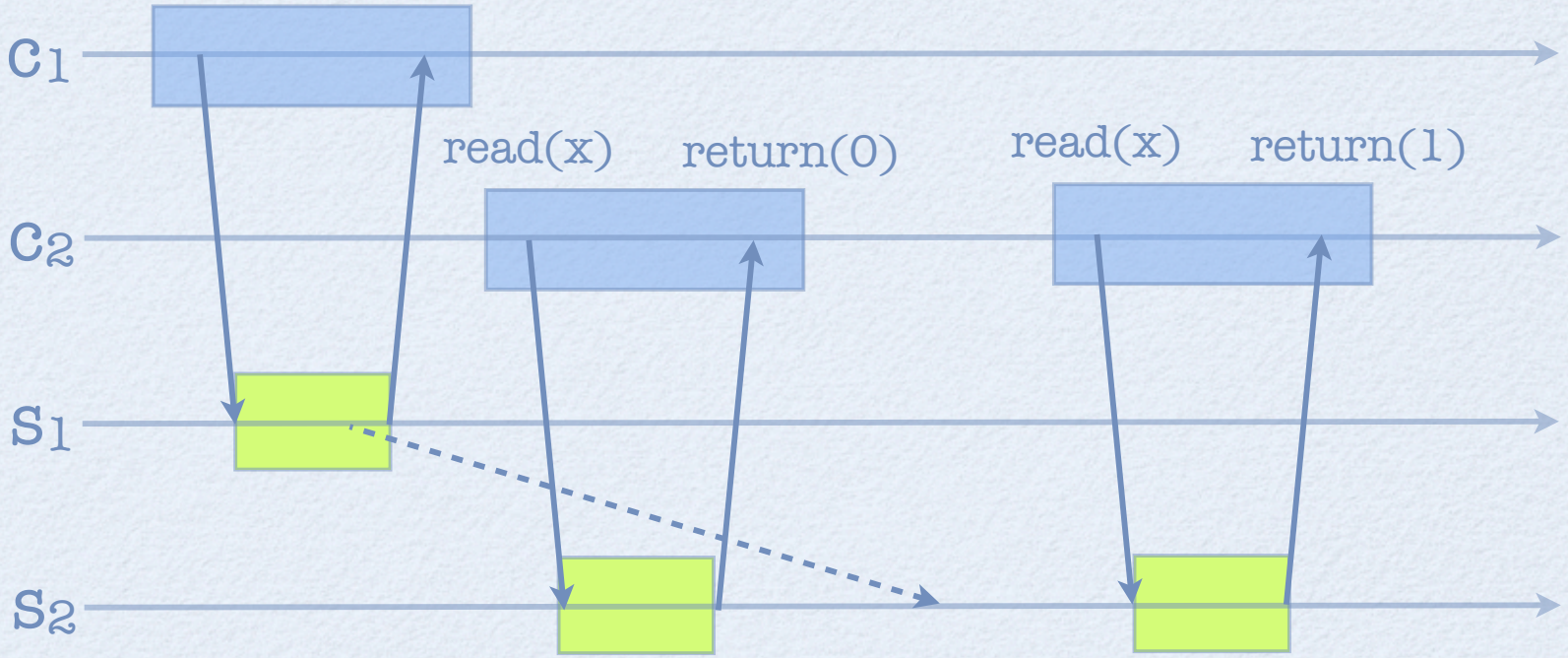
Replication model

- Consistency criteria
  - Multiple clients, multiple servers



Replication model

- Consistency criteria
  - Ideally, defines system behavior regardless of implementation details and the operation semantics
  - Is the following execution intuitive to clients?



Replication model

- Consistency criteria
  - Two consistency criteria for objects (among several others)
    - Linearizability
    - Sequential consistency

Replication model

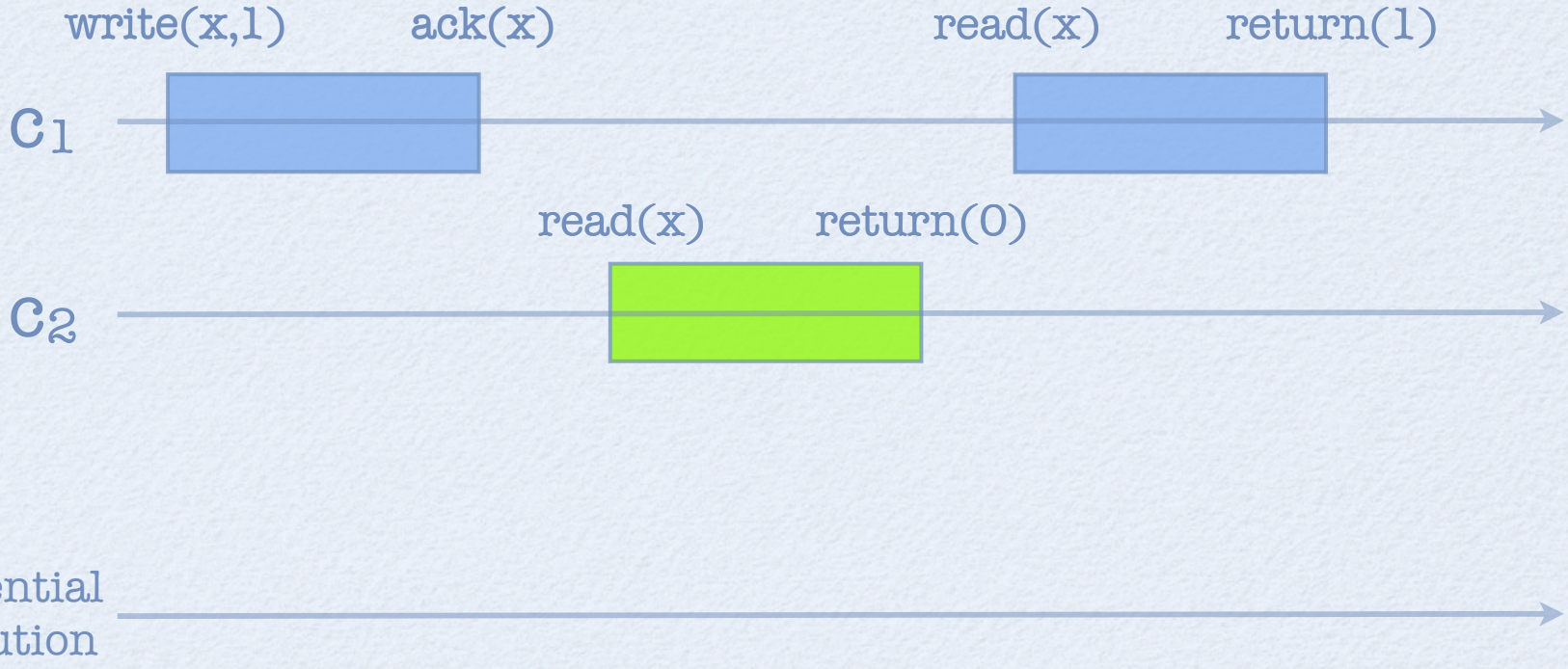
- Linearizability
  - A concurrent execution is linearizable if there is a sequential way to reorder the client operations such that:

    (1) it respects the semantics of the objects, as determined in their sequential specs

    (2) it respects the order of non-overlapping operations among all clients

Replication model

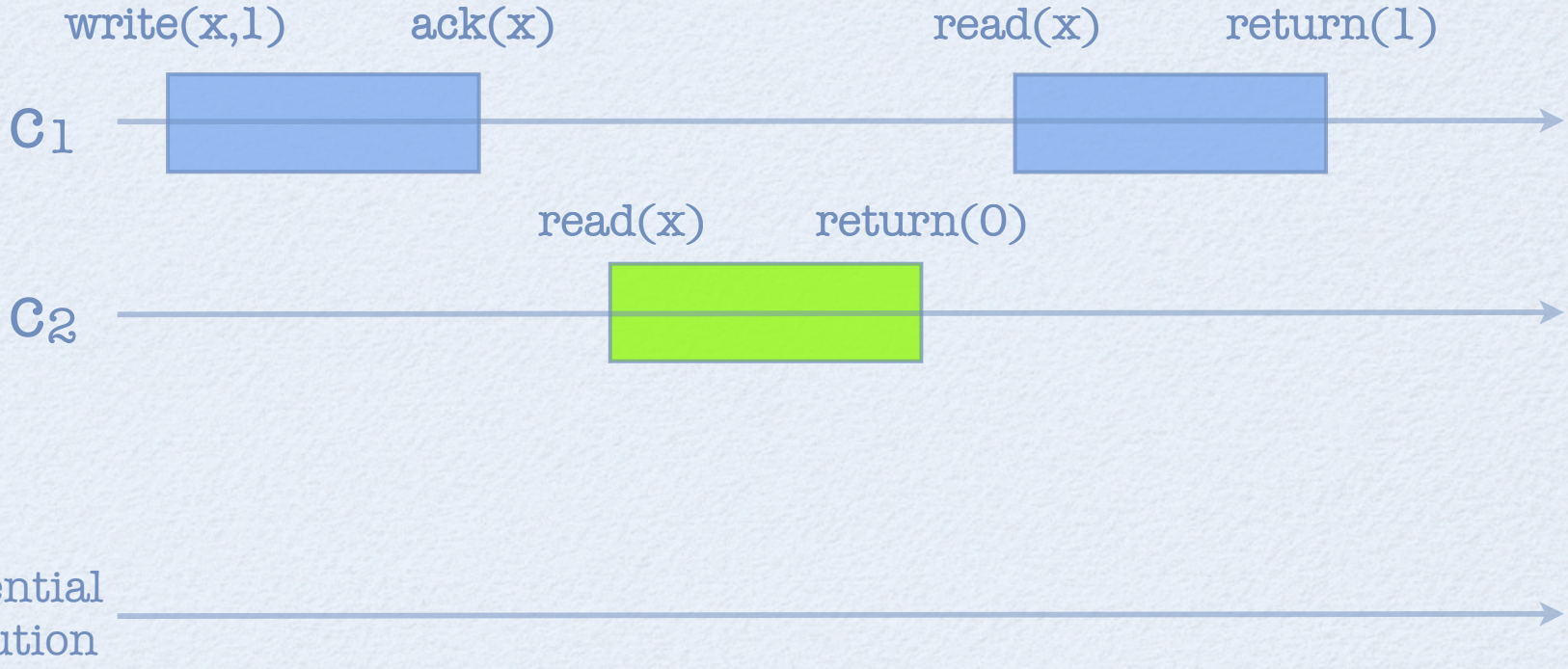
- Linearizability



Problem: Does not respect the semantics of the object

Replication model

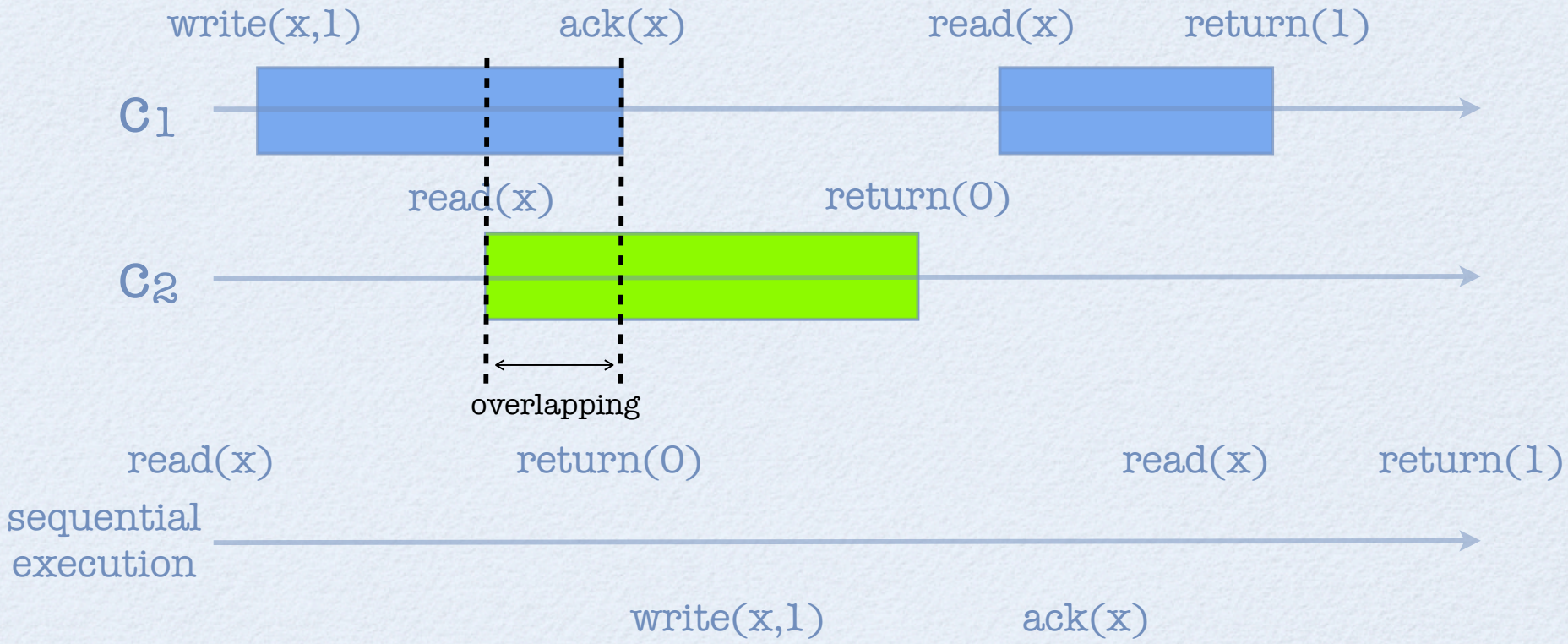
- Linearizability



Problem: Does not respect the order or non-overlapping operations

Replication model

- Linearizability



Linearizable execution!

Replication model

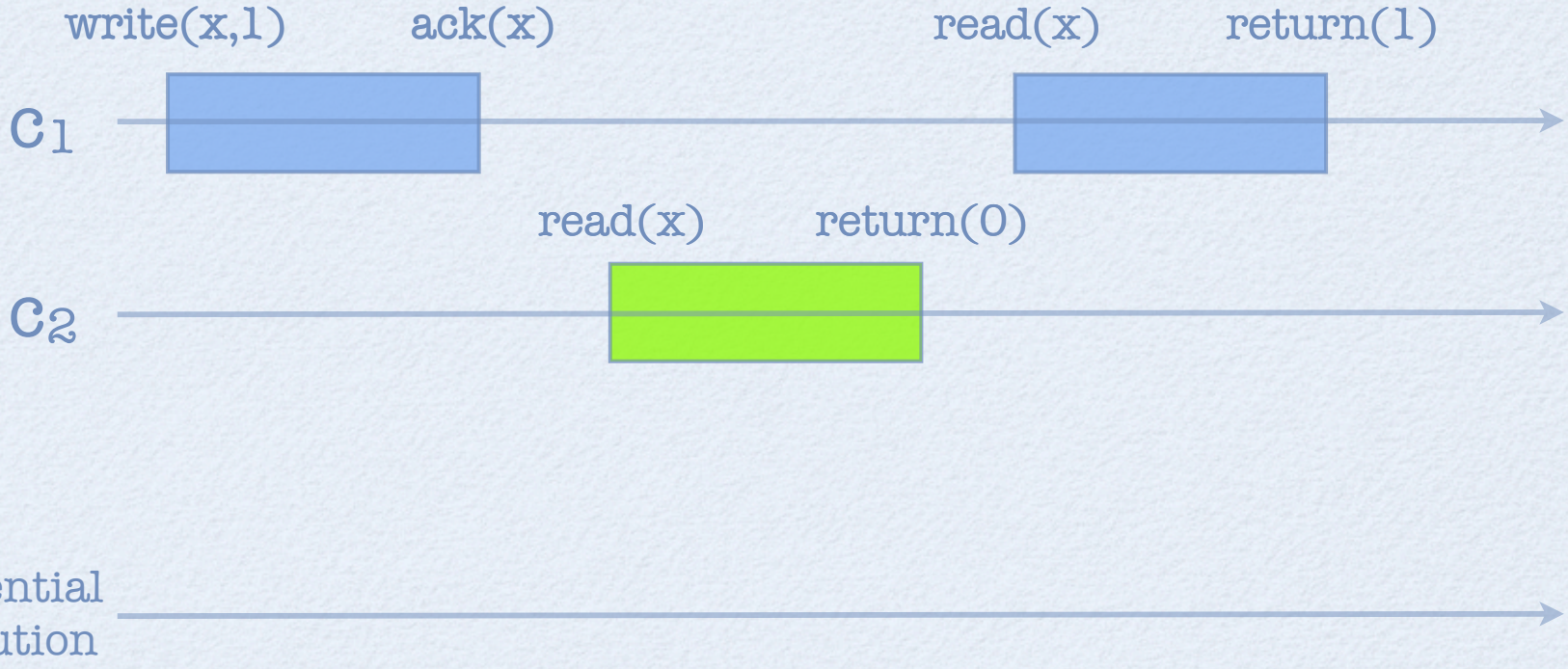
- Sequential consistency
  - A concurrent execution is sequentially consistent if there is a sequential way to reorder the client operations such that:

    (1) it respects the semantics of the objects, as determined in their sequential specs

    (2) it respects the order of operations at the client that issued the operations

Replication model

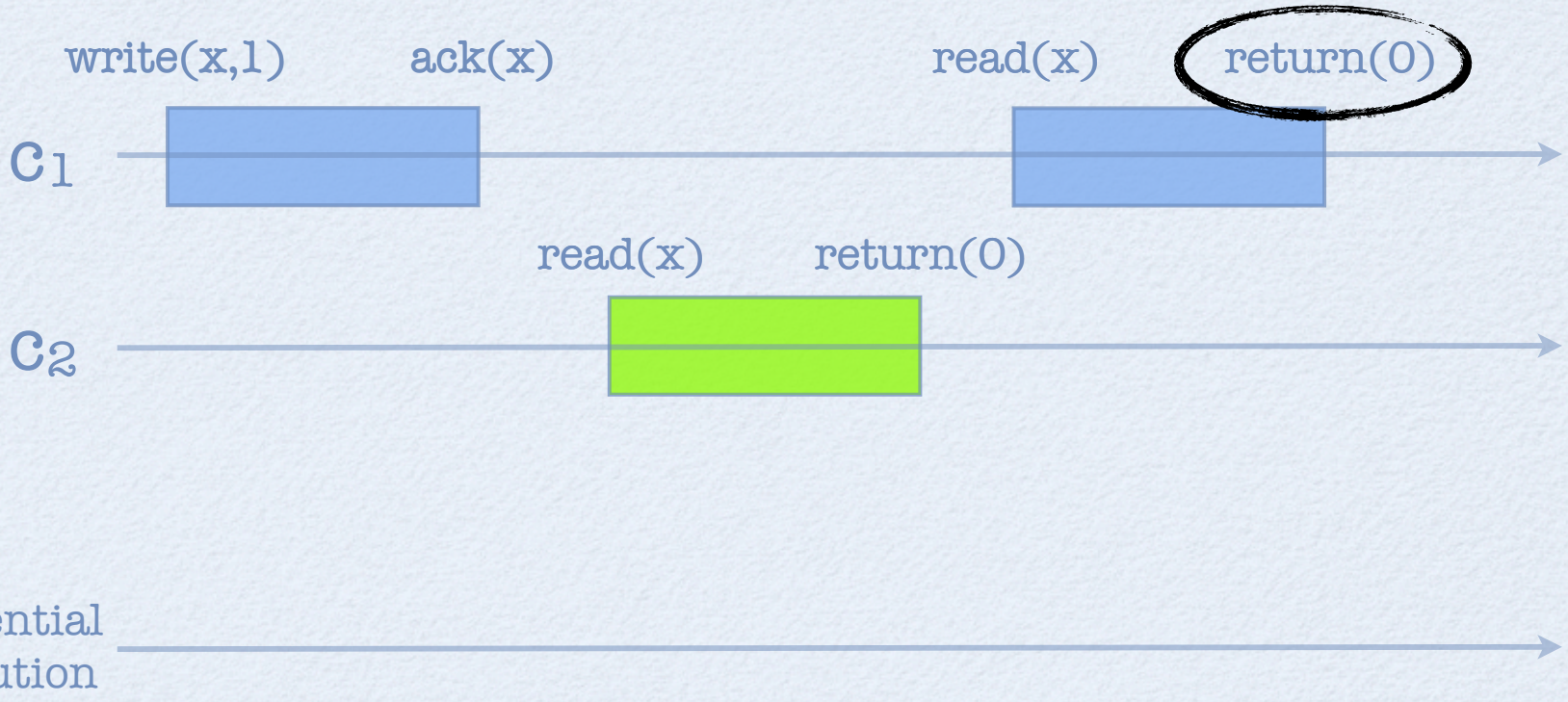
- Sequential consistency



Sequentially consistent!

Replication model

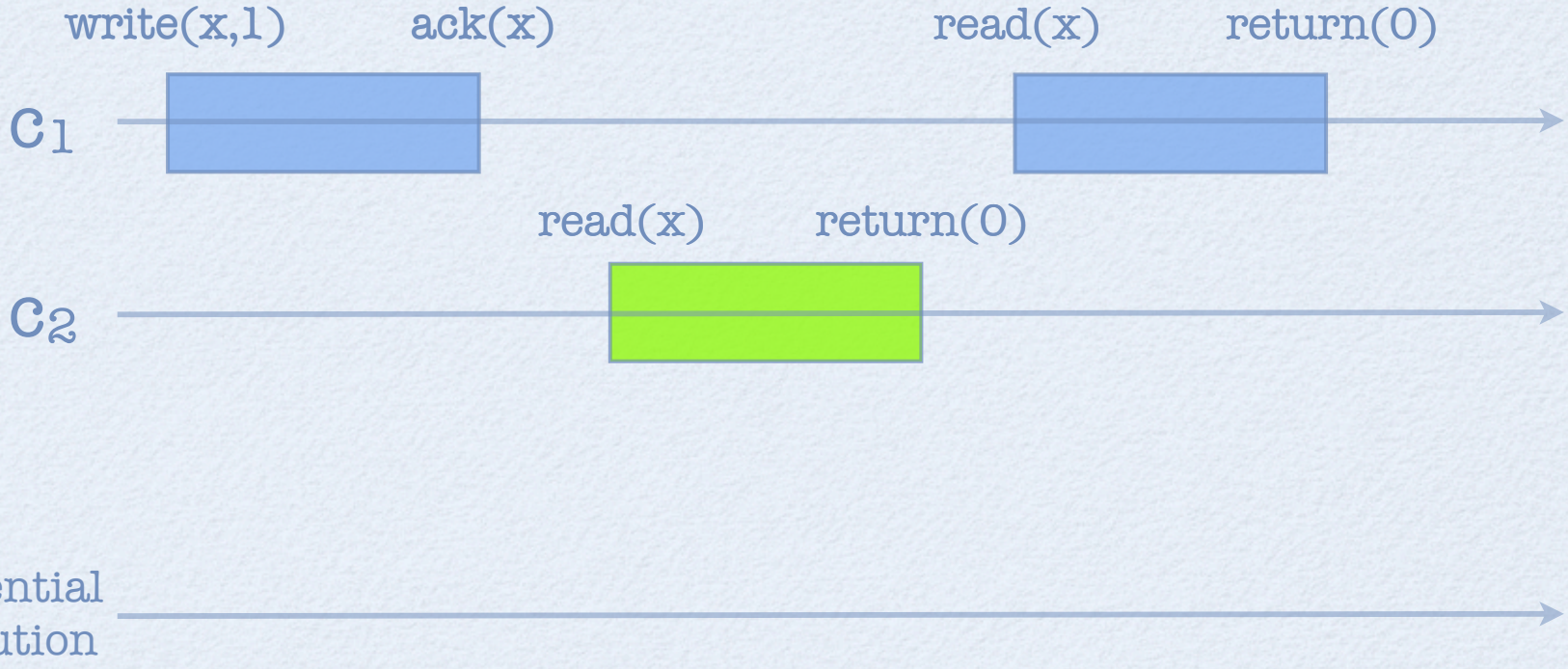
- Sequential consistency



Problem: Does not respect the order or non-overlapping operations

Replication model

- Sequential consistency



Problem: Does not respect the order or operations at c_1

Replication model

- **Database model**
  - Clients: c_1, c_2,...
  - Servers: s_1, ..., s_n
  - Operations: **read**, **write**, **commit** and **abort**
  - Transaction: group of read and/or write operations followed by commit or abort
    - ACID properties (see next)

Replication model

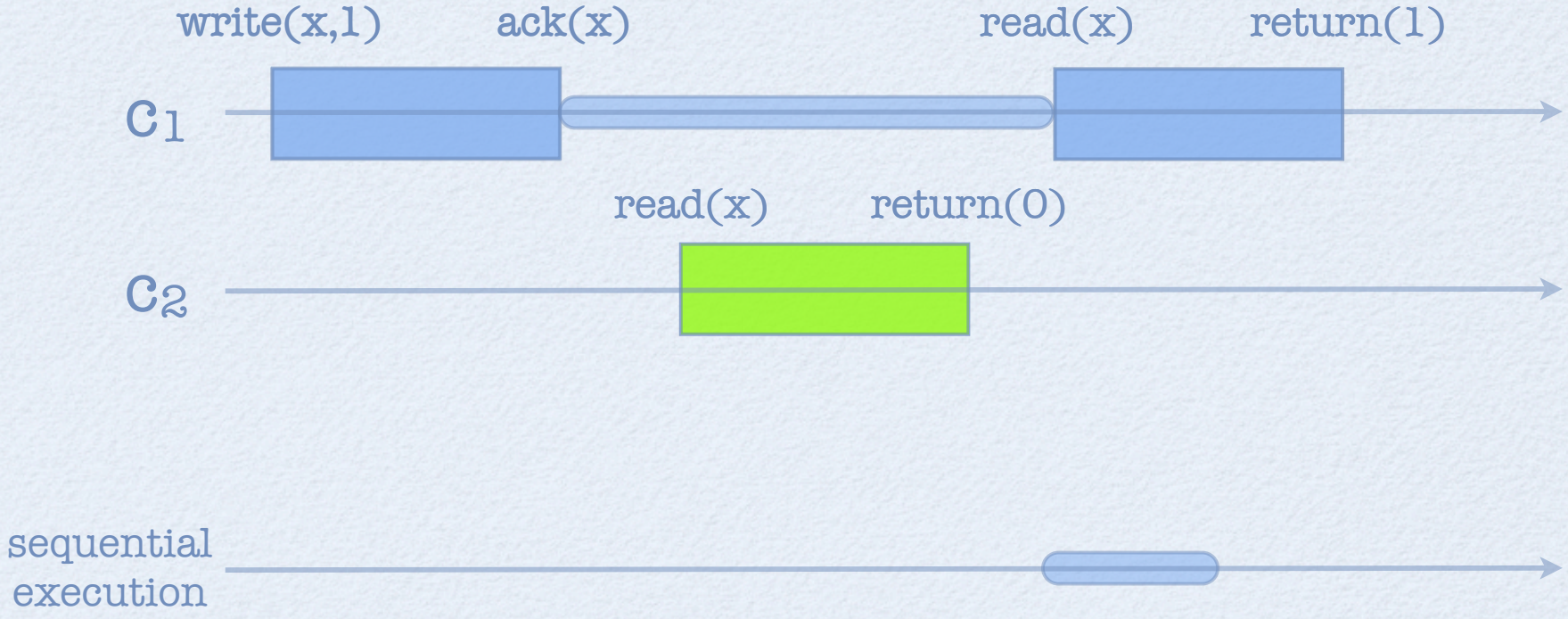
- Transaction properties
  - **A**tomicity: Either all transaction operations are executed or none are executed
  - **C**onsistency: A transaction is a correct transformation of the state
  - **I**solation: Serializability (next slide)
  - **D**urability: Once a transaction commits, its changes to the state survive failures

Replication model

- Serializability (1-copy SR)
  - A concurrent execution is serializable if it is equivalent to a serial execution with the same transactions
  - Two executions are (view) equivalent if:
    - Their transactions preserve the same “reads-from” relationships (t reads x from t’ in both executions)
    - They have the same final writes

Replication model

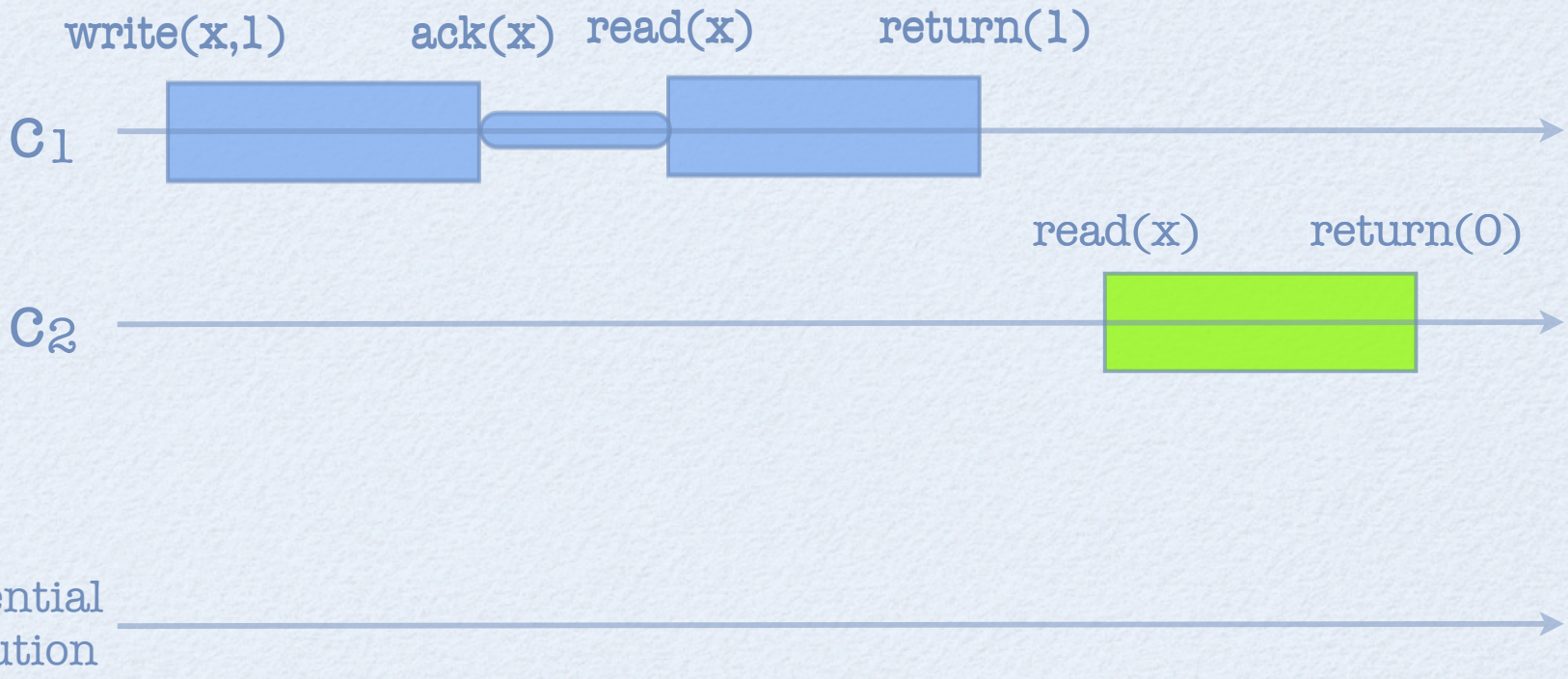
- Serializability



Serializable!

Replication model

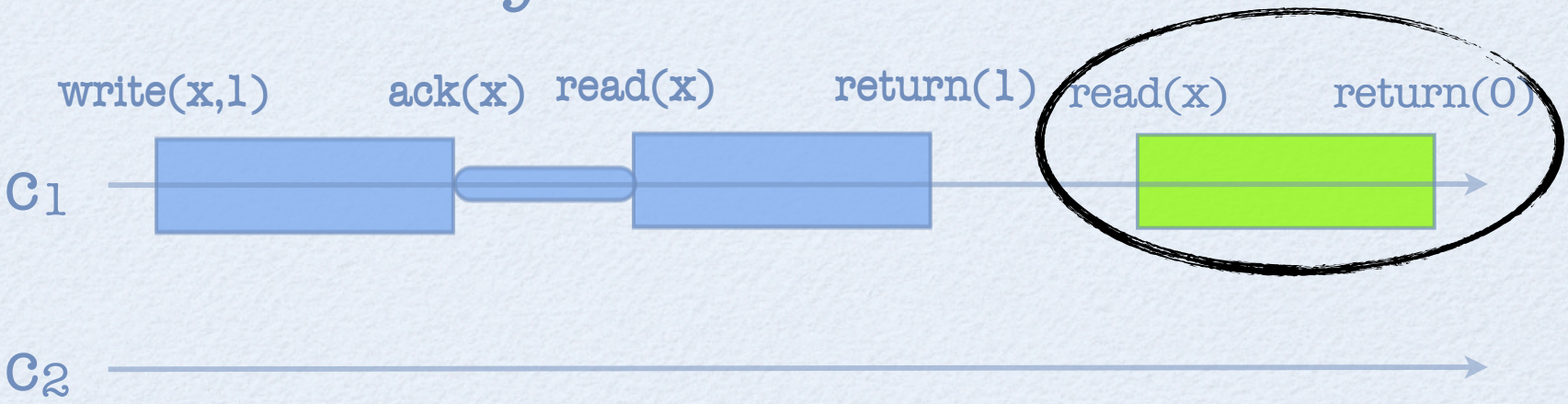
- Serializability



Serializable!

Replication model

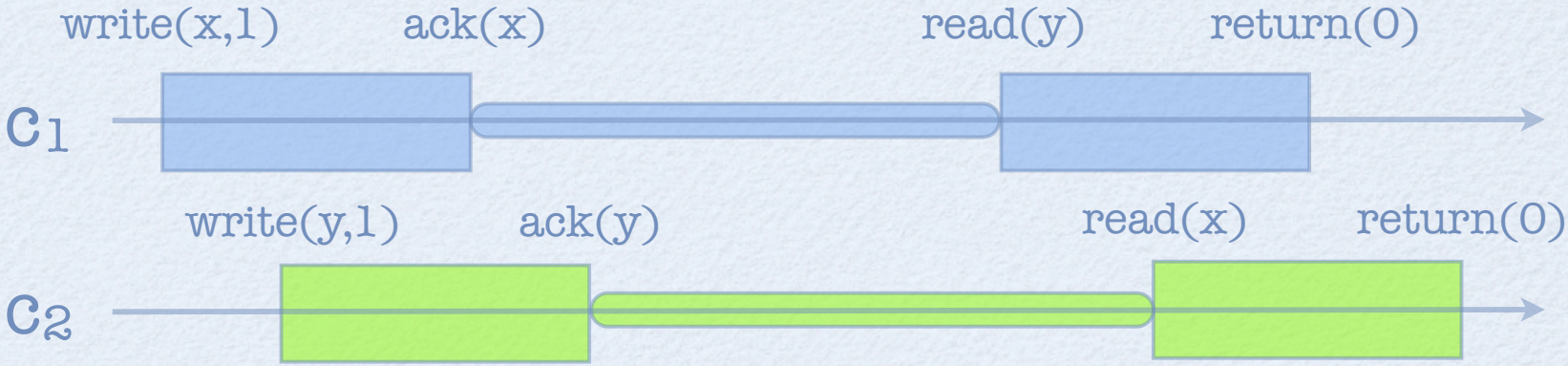
- Serializability



Serializable!!!

Replication model

- Serializability



Non serializable!

Replication model

- Object and database consistency criteria
  - Assume single-operation transactions

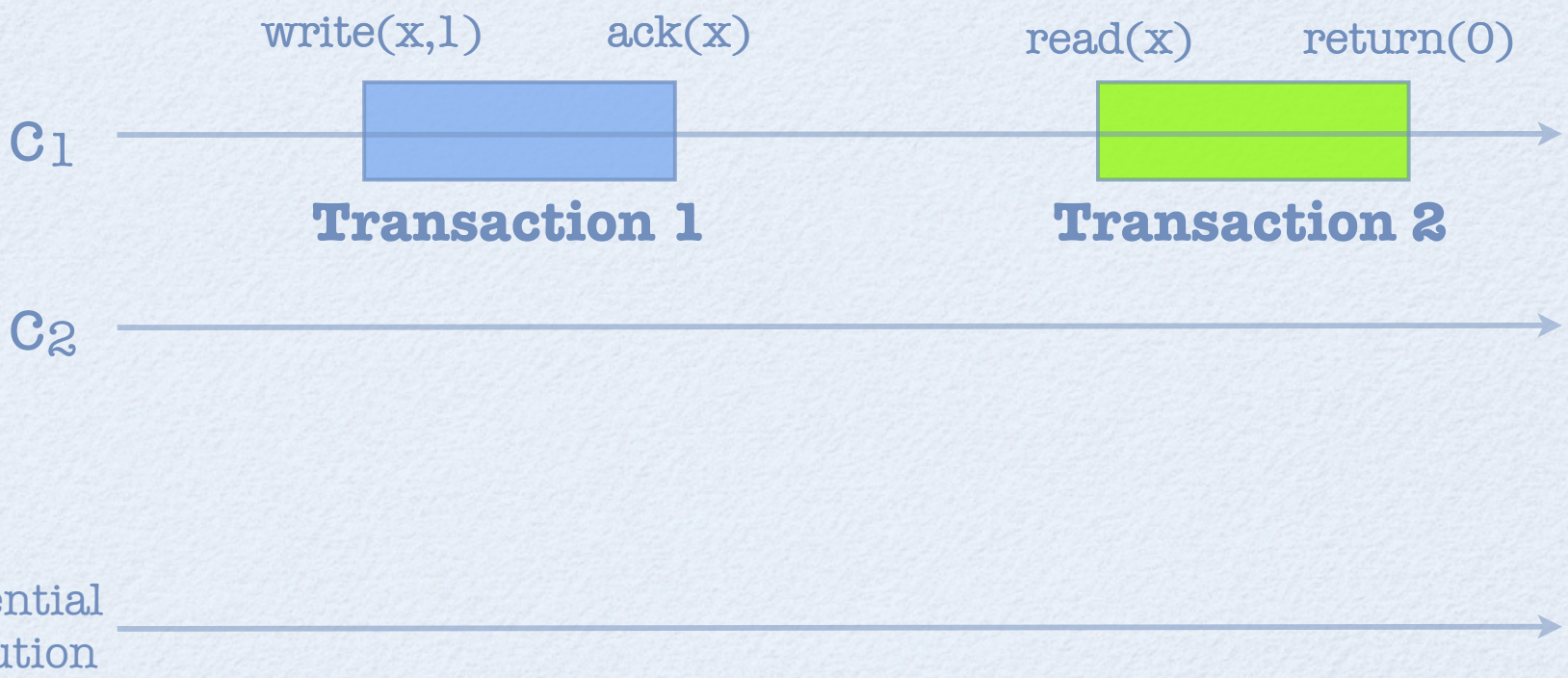
Objects	Database
Linearizability	Strong Serializability
Sequential Consistency	Session Serializability
?	Serializability

equivalent if transactions have only one operation

equivalent if transactions have only one operation

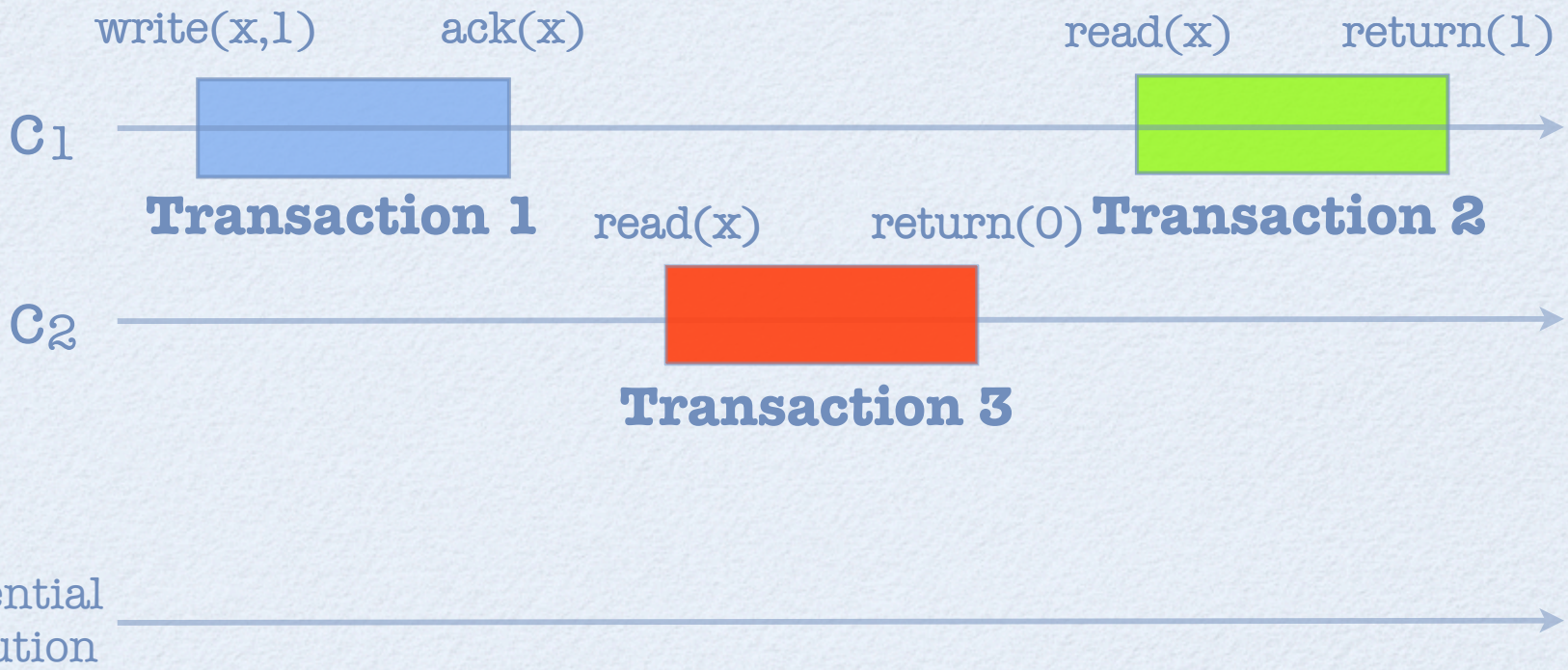
Replication model

- Serializable, not session-serializable



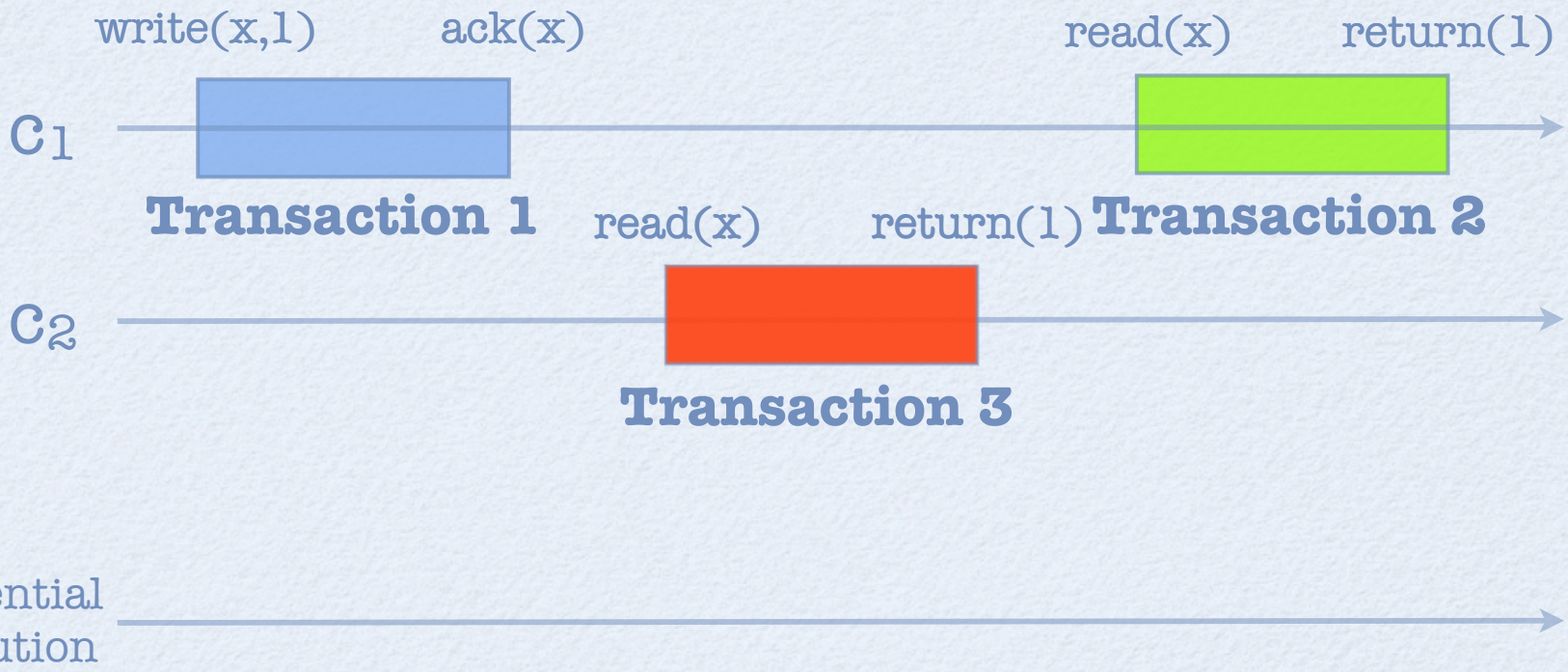
Replication model

- Session-serializable, not strongly serializable



Replication model

- Strongly serializable



Outline

- Motivation
- Replication model
- **From objects to databases**
- Deferred update replication
- Final remarks

From objects to databases

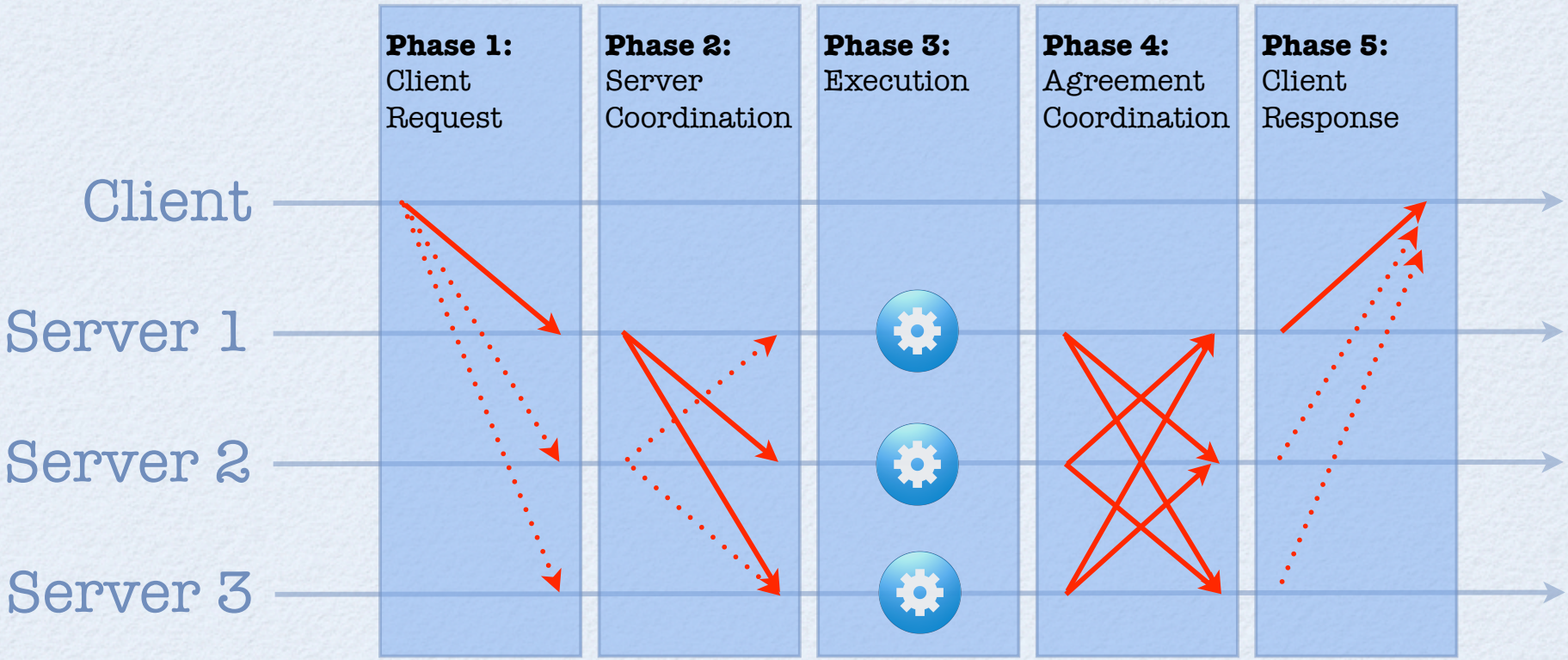
- Fundamentals
  - Model considerations
  - Generic functional model
- Object replication
  - Passive replication (primary-backup)
  - Active replication (state-machine replication)
  - Multi-primary passive replication

From objects to databases

- Model considerations
  - Client and server processes
  - Communication by message passing
  - Crash failures only
    - No Byzantine failures

Replication model

- Generic functional model

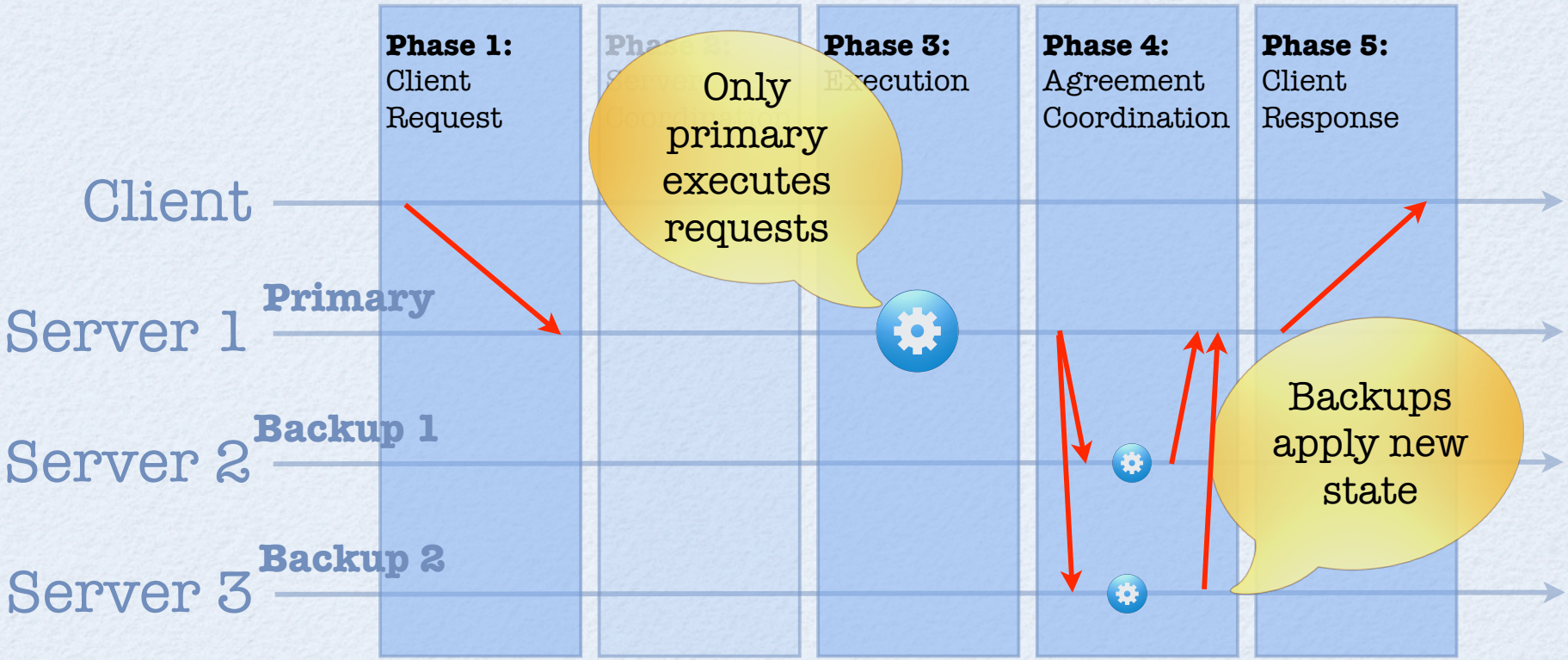


From objects to databases

- Passive replication
  - aka Primary-backup replication
  - Fail-stop failure model
    - a process follows its spec until it crashes
    - a crash is detected by every correct process
    - no process is suspected of having crashed until after it actually crashes
  - Algorithm ensures linearizability

From objects to databases

- Passive replication algorithm



From objects to databases

- Passive replication (Linearizability)
  - Perfect failure detection ensures that at most one primary exists at all times
  - A failed primary is detected by the backups
  - Eventually one backup replaces failed primary

From objects to databases

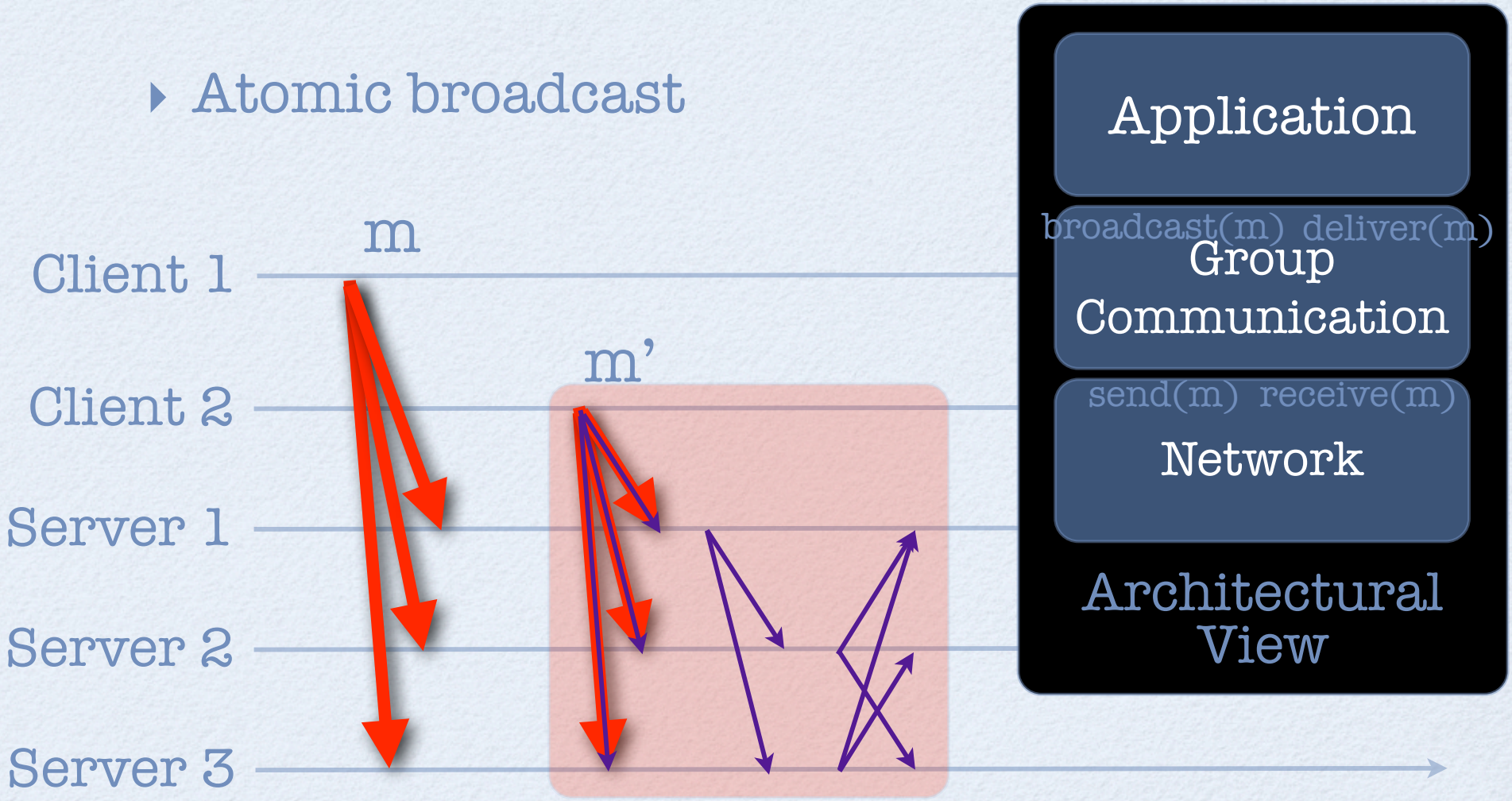
- Active replication
  - aka State-machine replication
  - Crash failure model
    - a process follows its spec until it crashes
    - a crash is detected by every correct process
    - correct processes may be erroneously suspected
  - Algorithm ensures linearizability

From objects to databases

- Atomic broadcast
  - Group communication abstraction
  - Primitives: **broadcast**(m) and **deliver**(m)
  - Properties
    - **Agreement**: Either all servers deliver m or no server delivers m
    - **Total order**: Any two servers deliver messages m and m' in the same order

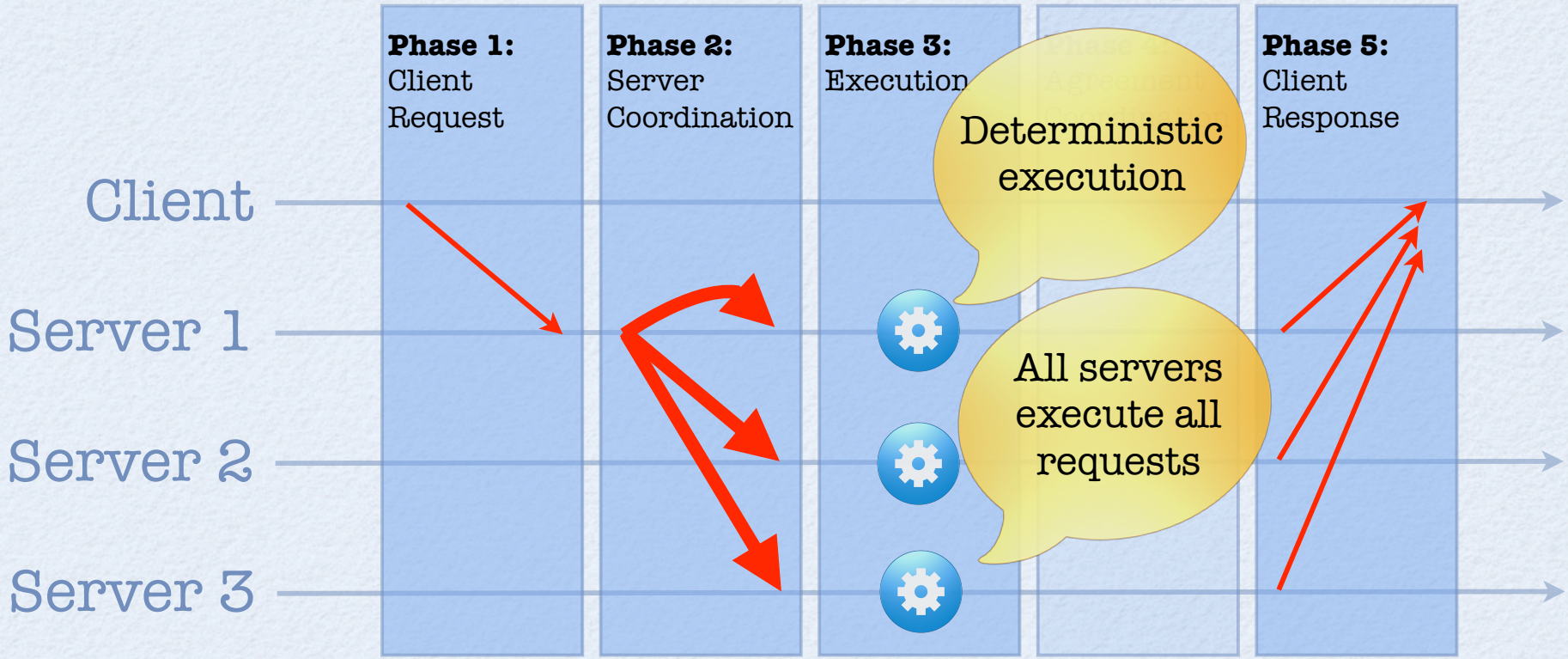
From objects to databases

- Atomic broadcast



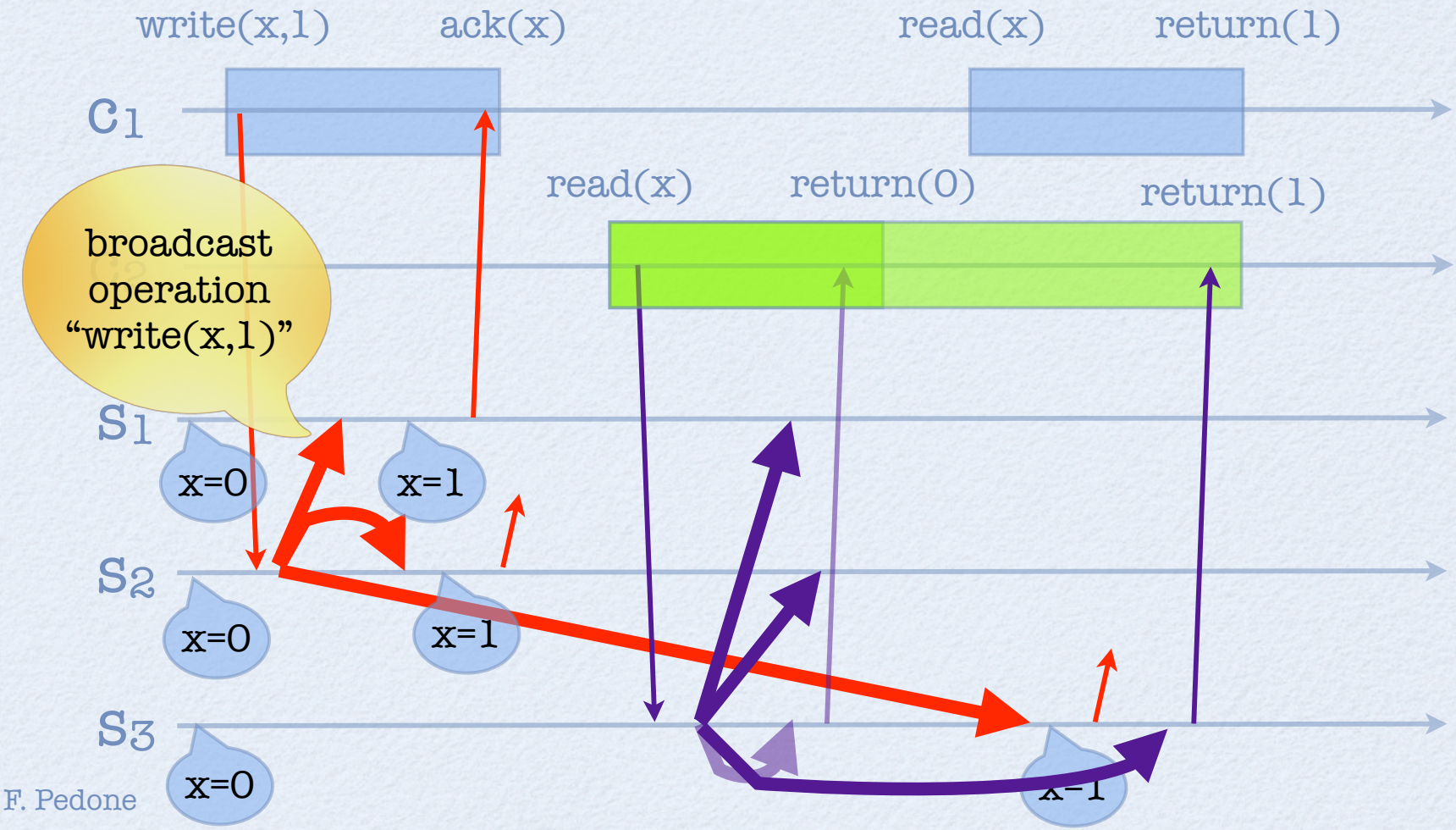
From objects to databases

- Active replication algorithm



From objects to databases

- Why it works



From objects to databases

- Active and passive replication
  - Target fault tolerance only
  - Not good for performance
    - Active replication:
      All servers execute all requests
    - Passive replication:
      Only one server executes requests

From objects to databases

- Multi-primary passive replication
  - Targets both fault tolerance and performance
  - Does not require perfect failure detection
  - Same resilience as active replication, but…
  - Better performance
    - Distinguishes read from write operations
    - Only one server executes each request

From objects to databases

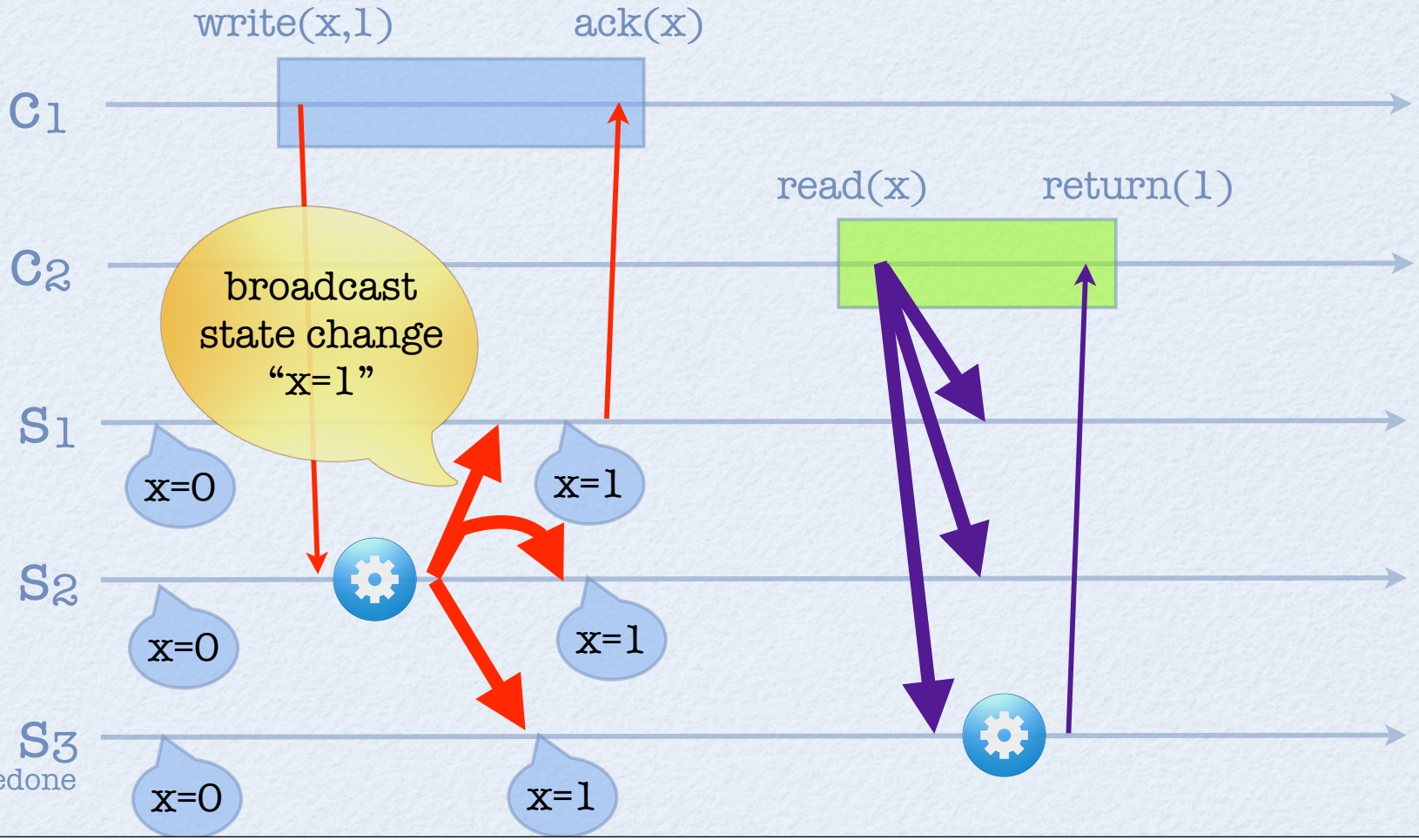
- Multi-primary passive replication
  - Read requests
    - Broadcast to all servers
    - Executed by only one server
    - Response is sent to the client

From objects to databases

- Multi-primary passive replication
  - Write requests
    - Executed by one server
    - State changes (“diff”) are broadcast to all servers
    - If the changes are “compatible” with the previous installed states, then a new state is installed, and the result of the request is returned to the client
    - Otherwise the request is re-executed

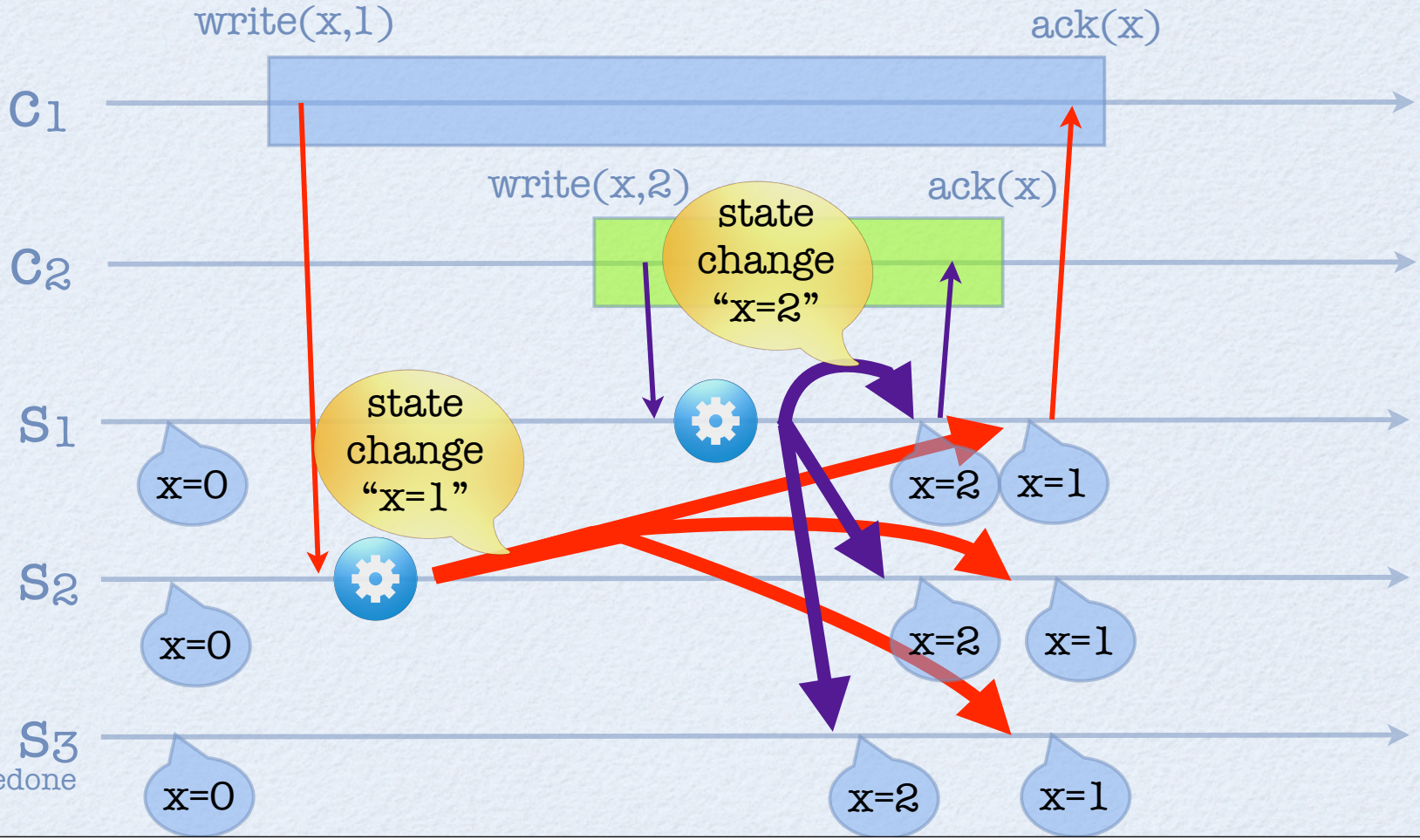
From objects to databases

- Multi-primary passive replication



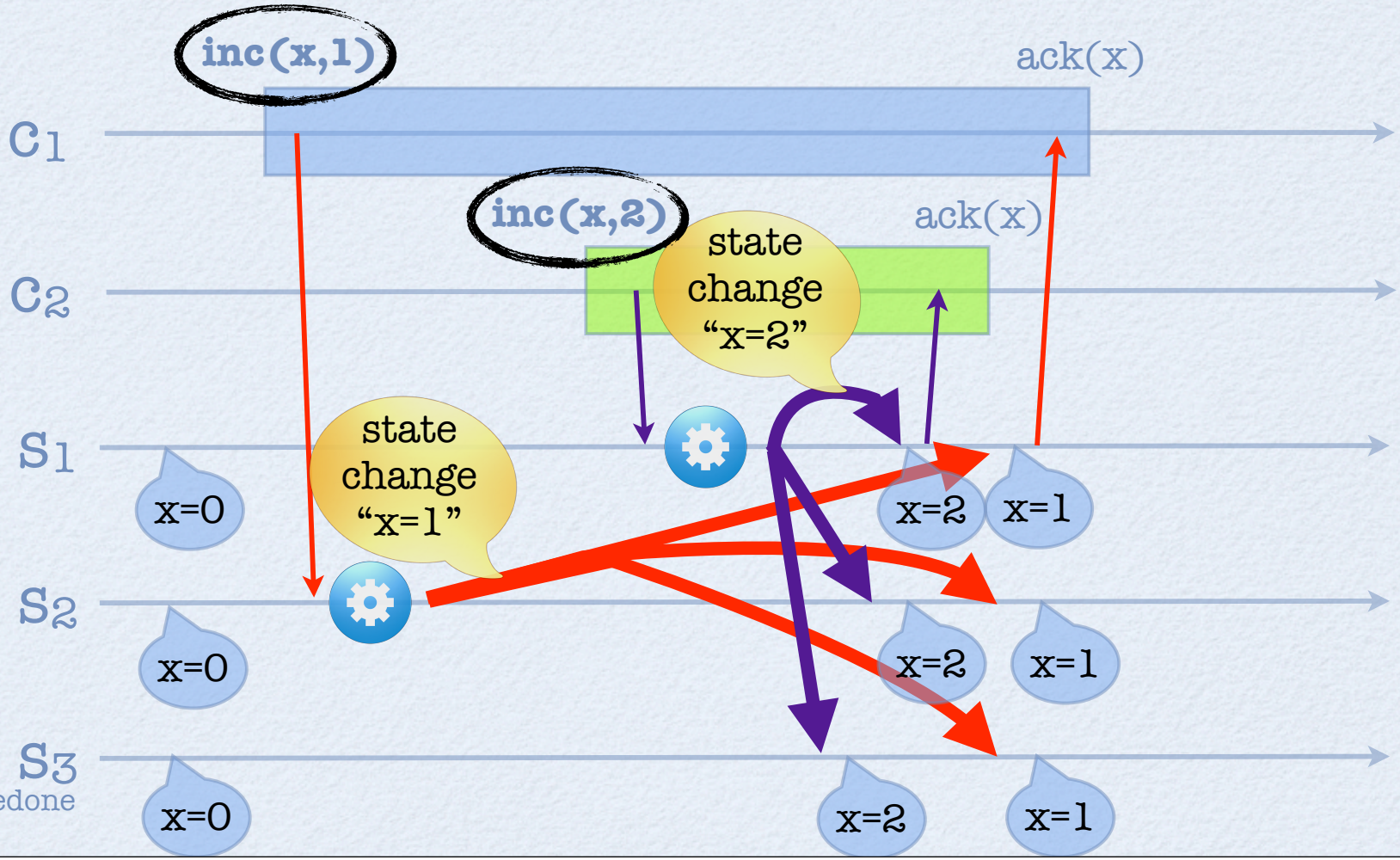
From objects to databases

- Multi-primary passive replication



From objects to databases

- Multi-primary passive replication

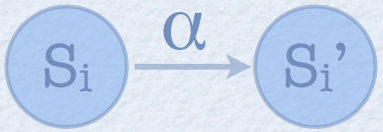


From objects to databases

- Multi-primary passive replication
  - Write requests
    - Executed by one server
    - State changes ("diff") are broadcast to all servers
    - **If the changes are "compatible" with the previous installed states, then a new state is installed, and the result of the request is returned to the client**
    - **Otherwise the request is re-executed**

From objects to databases

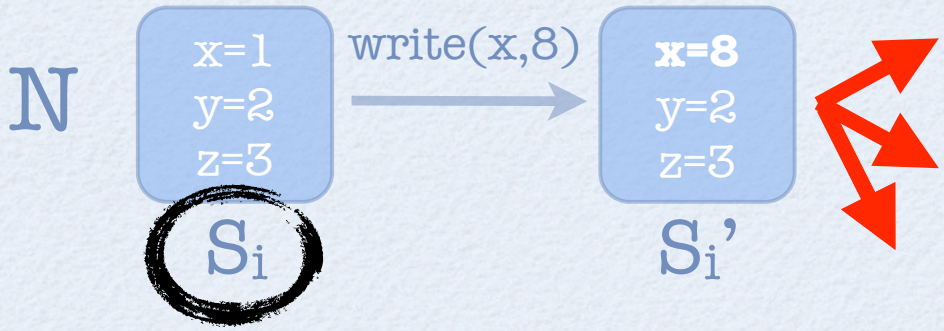
- Multi-primary passive replication
  - Compatible states

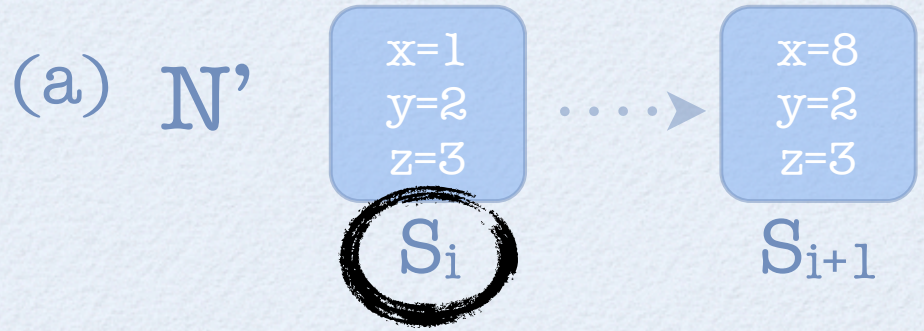


    - Let S_i be the state at server N before it executes operation α, and S_i' the state after α
    - Let $h=S_0 \circ \ldots \circ S_j$ be the sequence of installed states at a server N' when it tries to install the new state S_i'
    - S_i' is compatible with h if
      - (a) $S_j = S_i$ or
      - (b) α does not read any variables modified in $S_{i+1}, S_{i+2}, \ldots, S_j$

From objects to databases

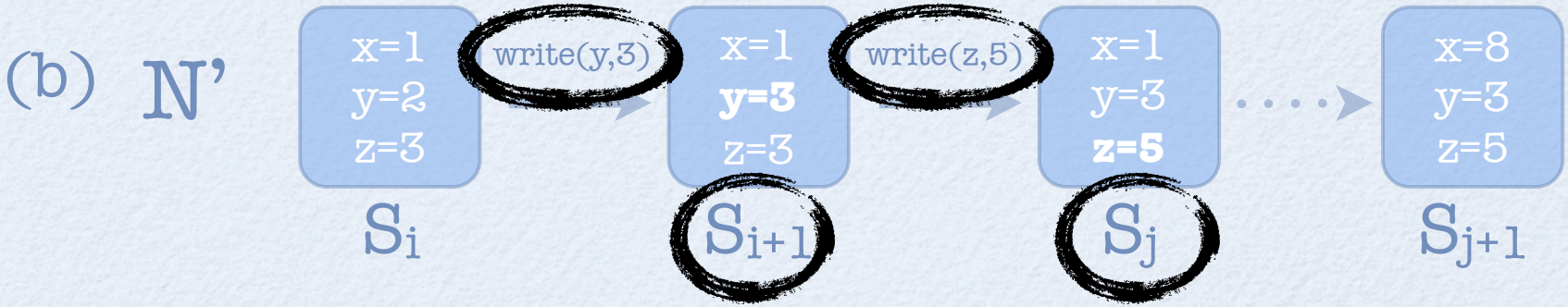
- Compatible states





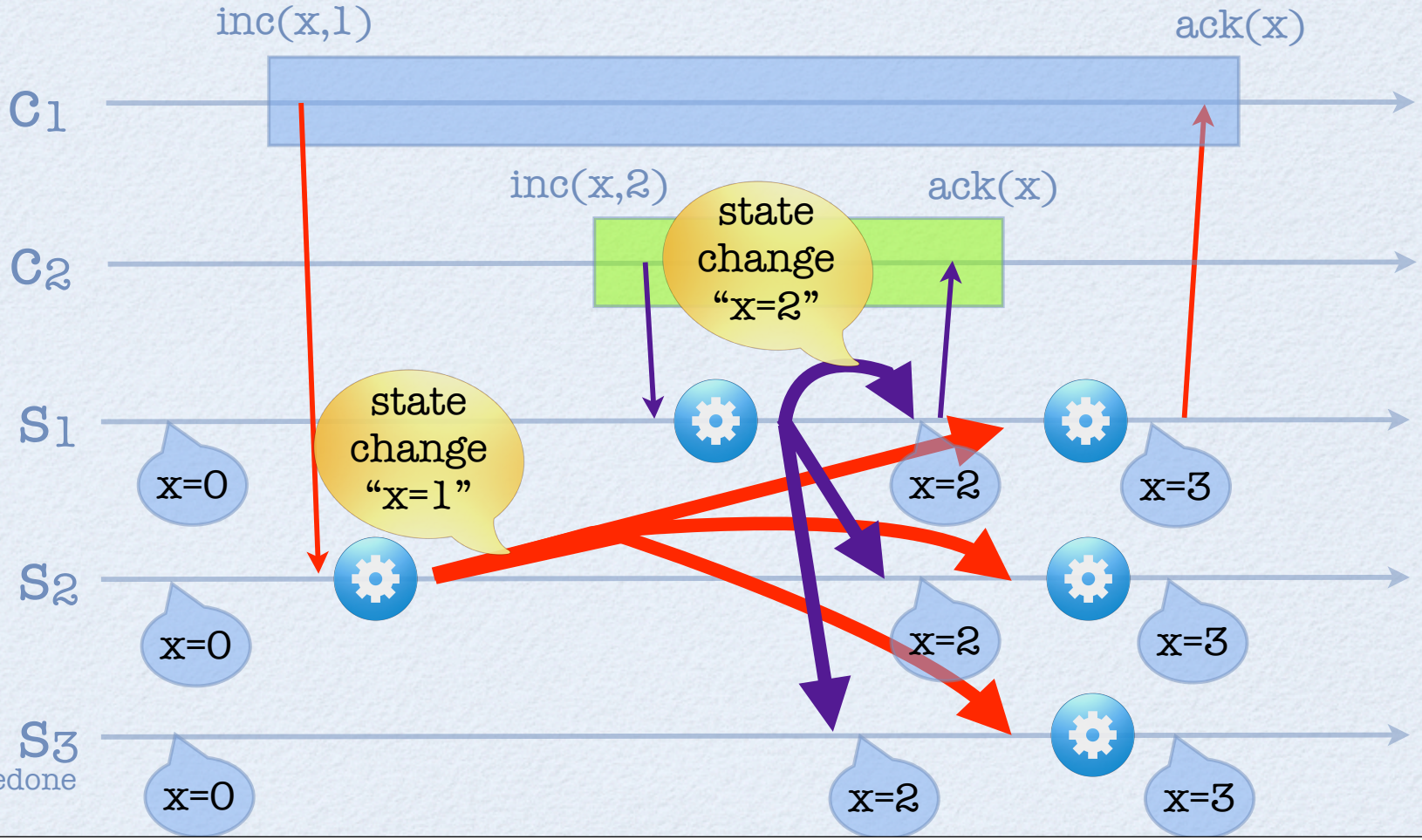
From objects to databases

- Compatible states



From objects to databases

- Multi-primary passive replication



From objects to databases

- Multi-primary passive replication
  - Optimistic approach
  - May or not rely on deterministic execution
    - It depends on how re-executions are dealt with
  - Installing a new state is cheaper than executing the request (that creates the state)

From objects to databases

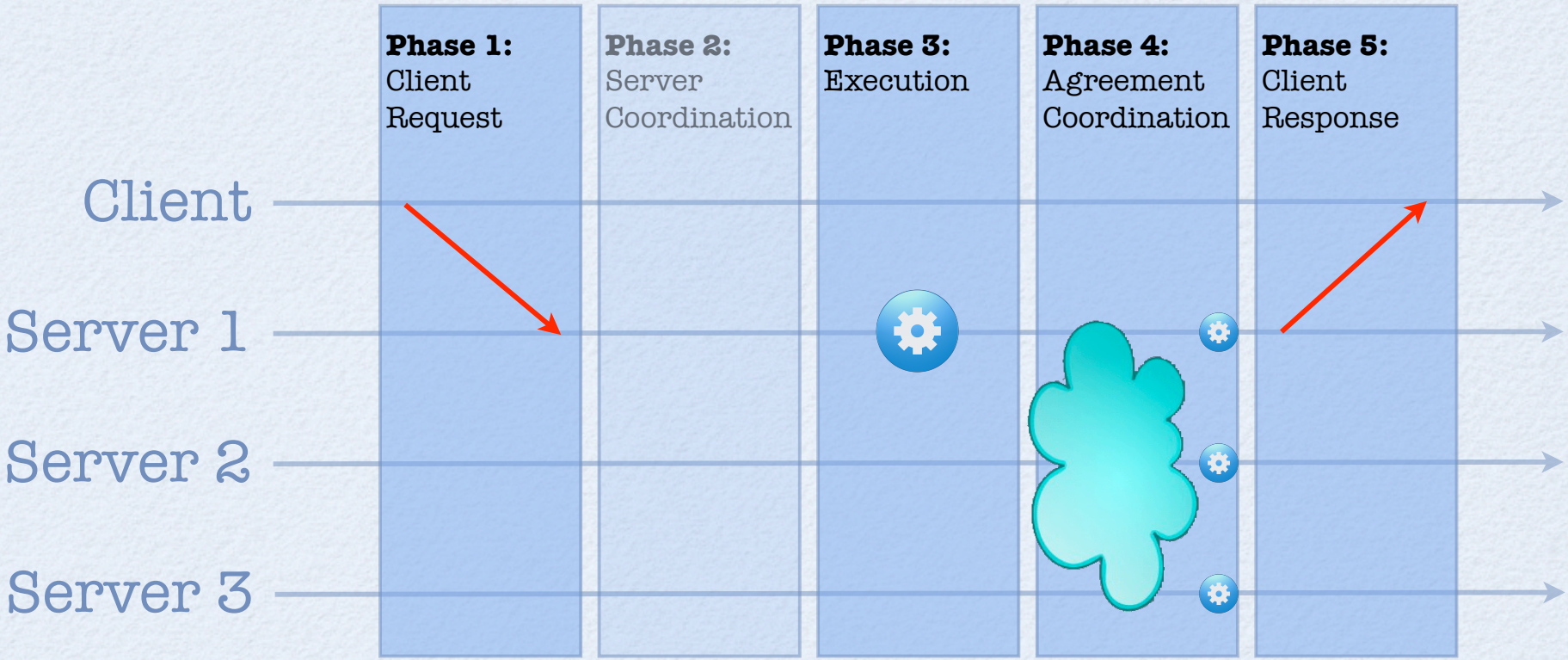
- Multi-primary passive replication
  - Provides both fault tolerance and performance
  - **Suitable for database replication!**

Outline

- Motivation
- Replication model
- From objects to databases
- **Deferred update replication**
- Final remarks

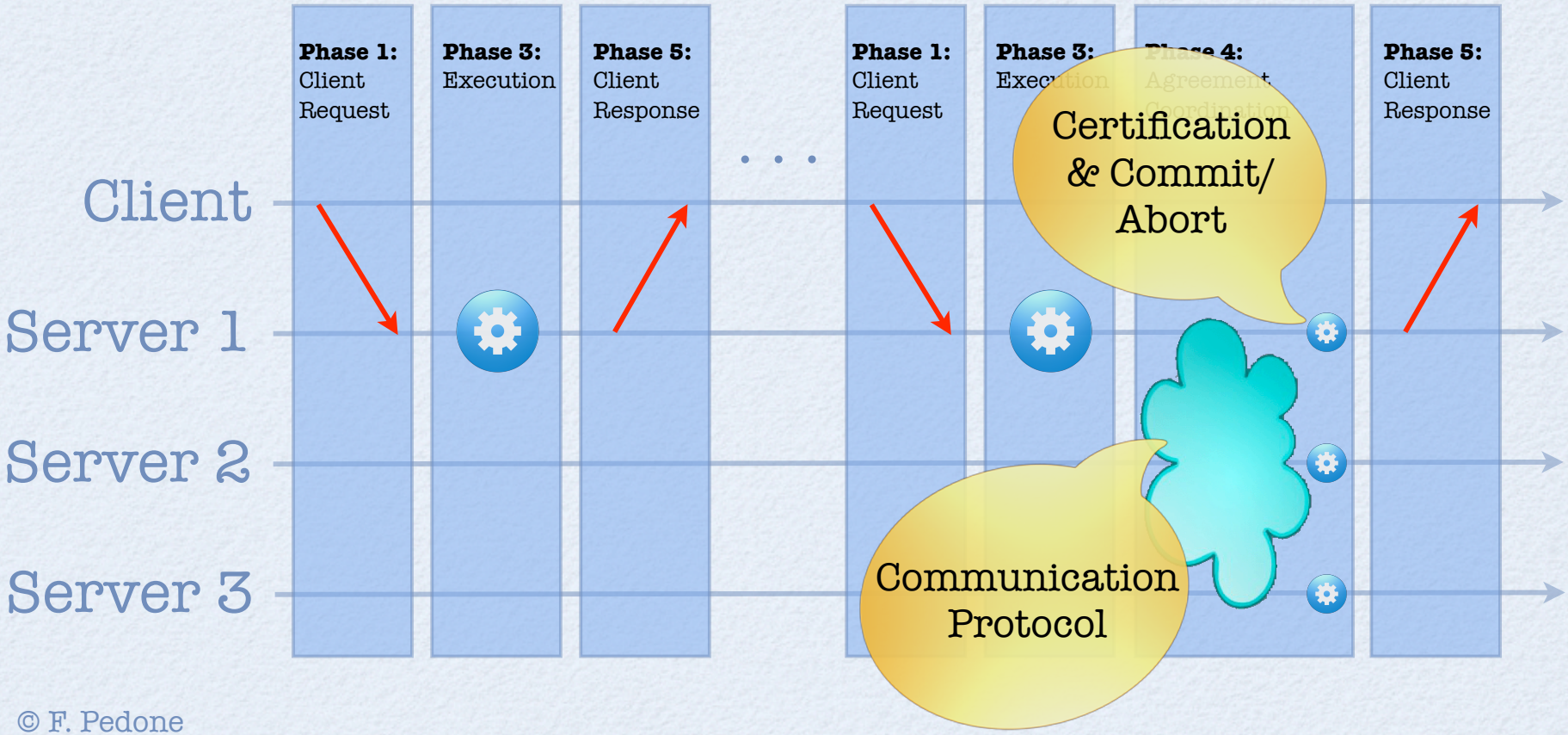
Deferred update replication

- General idea (I)



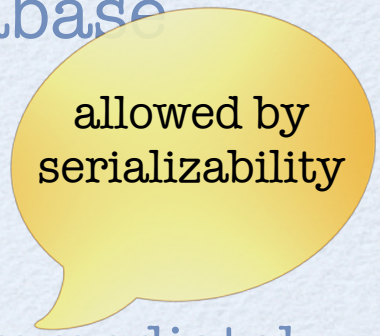
Deferred update replication

- General idea (II)



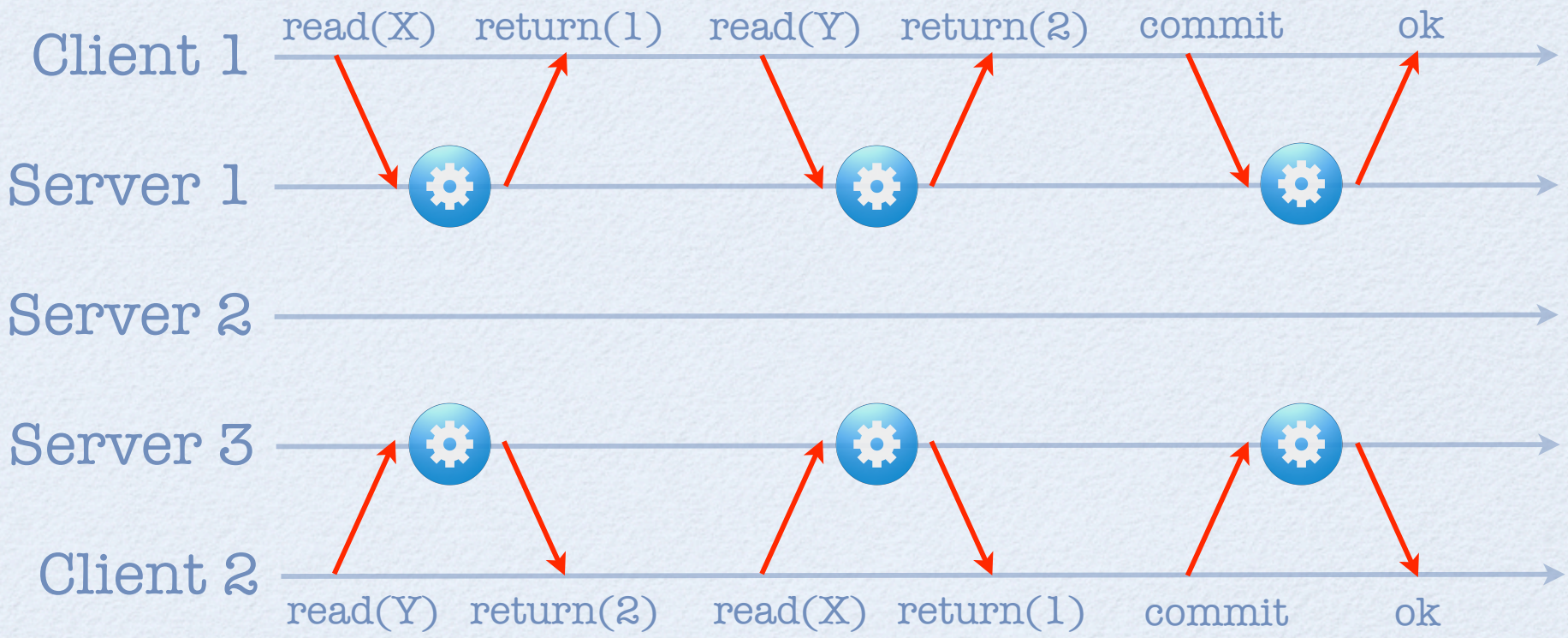
Deferred update replication

- The protocol in detail
  - Transactions (read-only and update) are submitted and executed by one database server
  - At commit time:
    - Read-only transactions are committed immediately
    - Update transactions are propagated to the other replicas for certification, and possibly commit



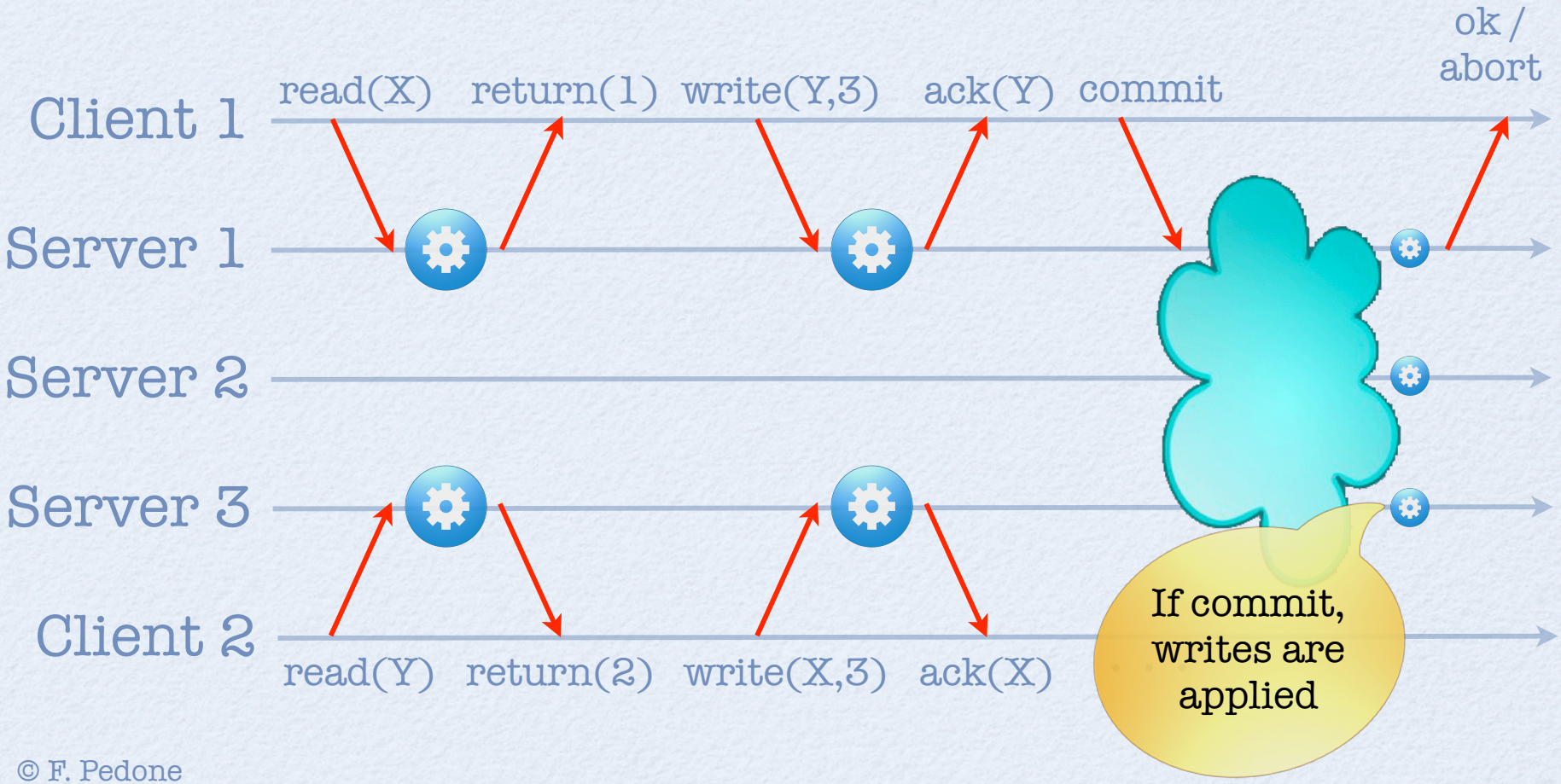
Deferred update replication

- Read-only transactions



Deferred update replication

- Update transactions

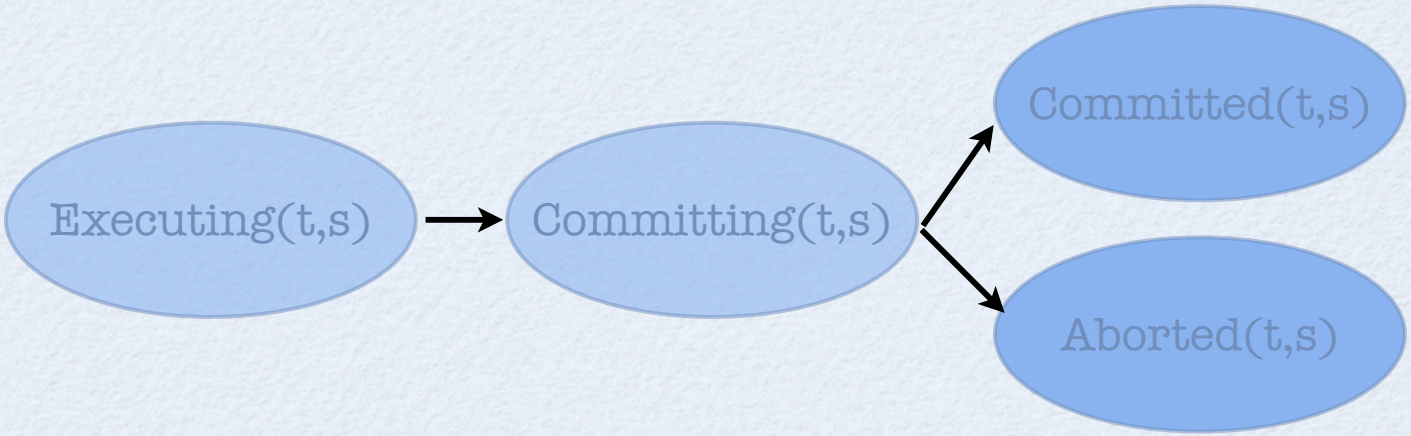


Deferred update replication

- Termination based on Atomic Broadcast
  - Similar to multi-primary passive replication
  - Deterministic certification test
  - Lower abort rate than atomic commit based technique

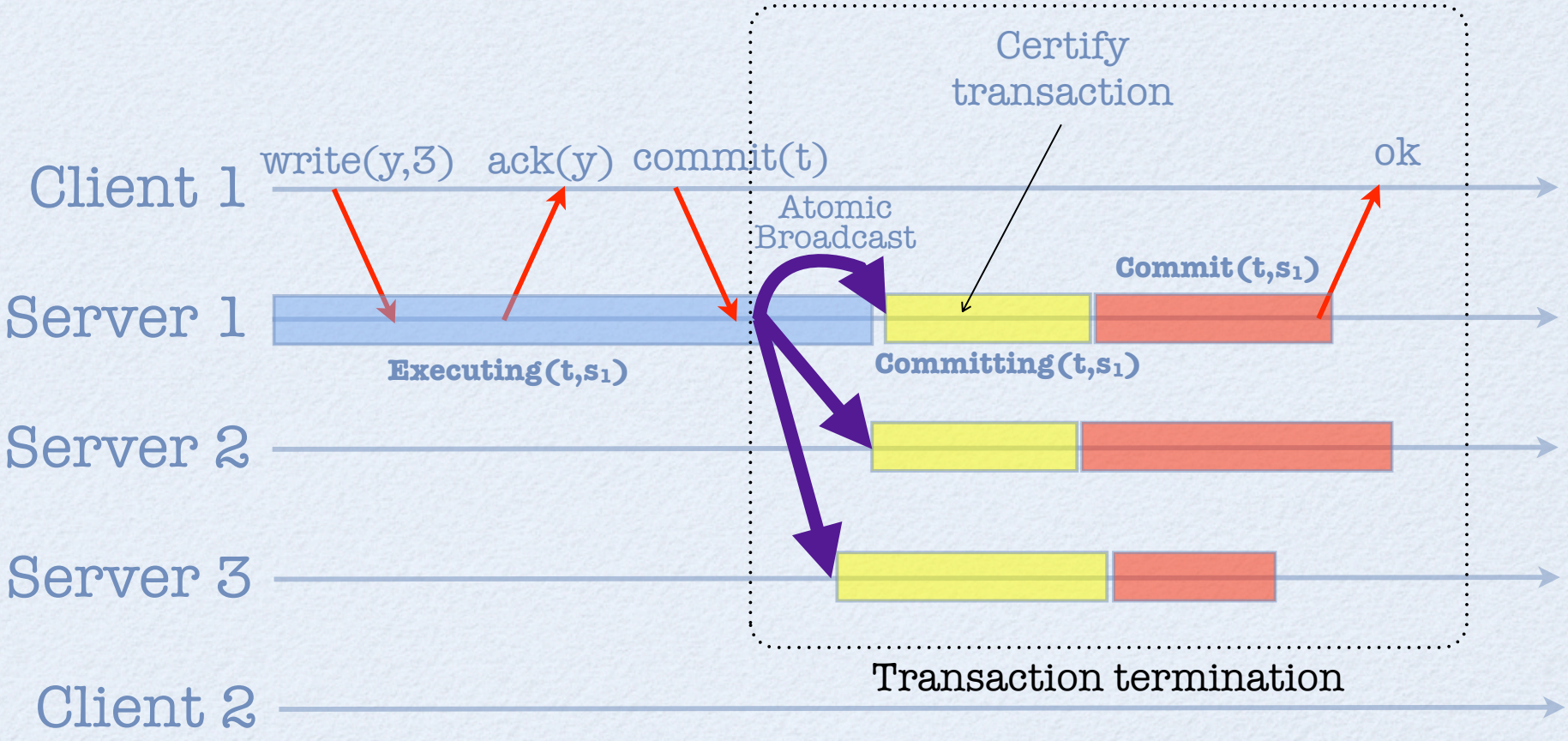
Deferred update replication

- Transaction states
  - Executing(t,s): transitory state
  - Committing(t,s): transitory state
  - Committed(t,s): final state
  - Aborted(t,s): final state



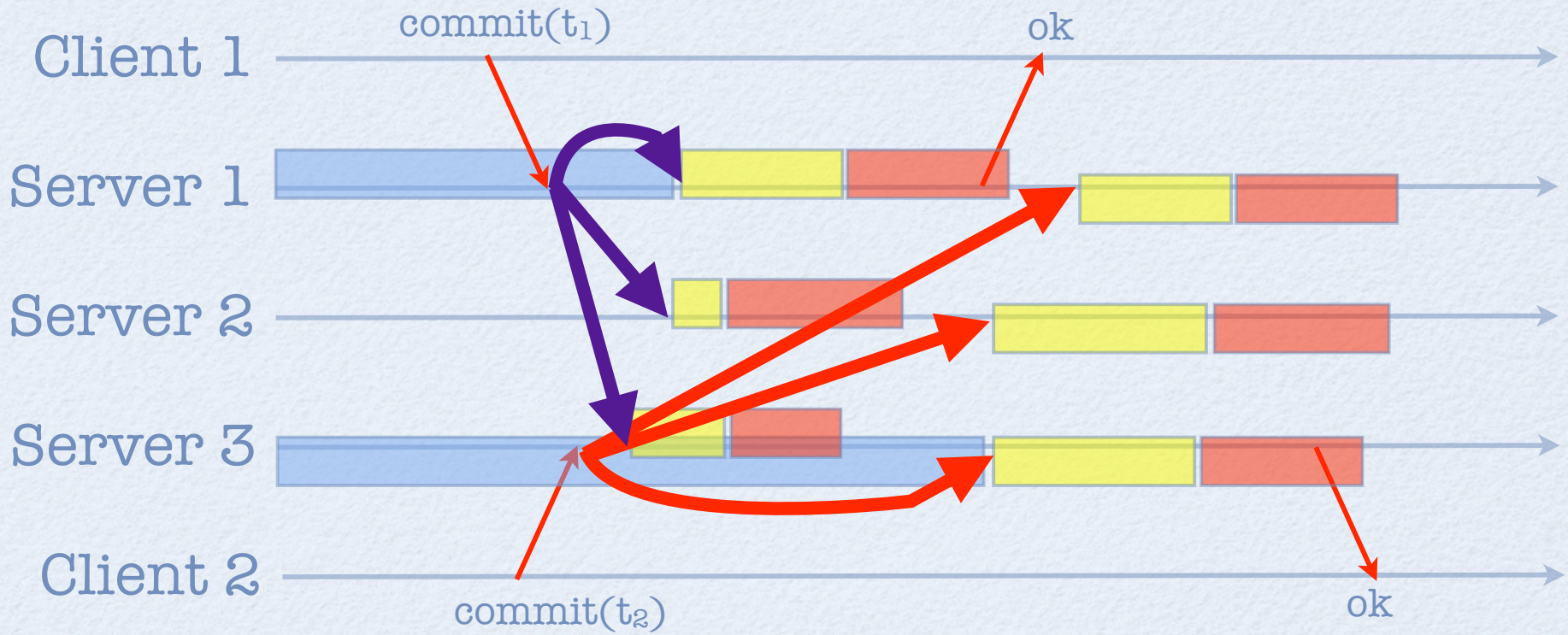
Deferred update replication

- Transaction states



Deferred update replication

- Transaction states



Deferred update replication

- Atomic broadcast-based certification

$$\forall t \, \forall s : Committing(t, s) \rightsquigarrow Committed(t, s) \equiv$$

$$\left[\begin{array}{l} \forall t' \text{ s.t. } Committed(t', s) : \\ \\ t' \rightarrow t \vee WS(t') \cap RS(t) = \emptyset \end{array} \right]$$

t'→t, t' precedes t: changes made by t' could be seen by t (i.e,. read by t)

t' does not update any item that t reads

Deferred update replication

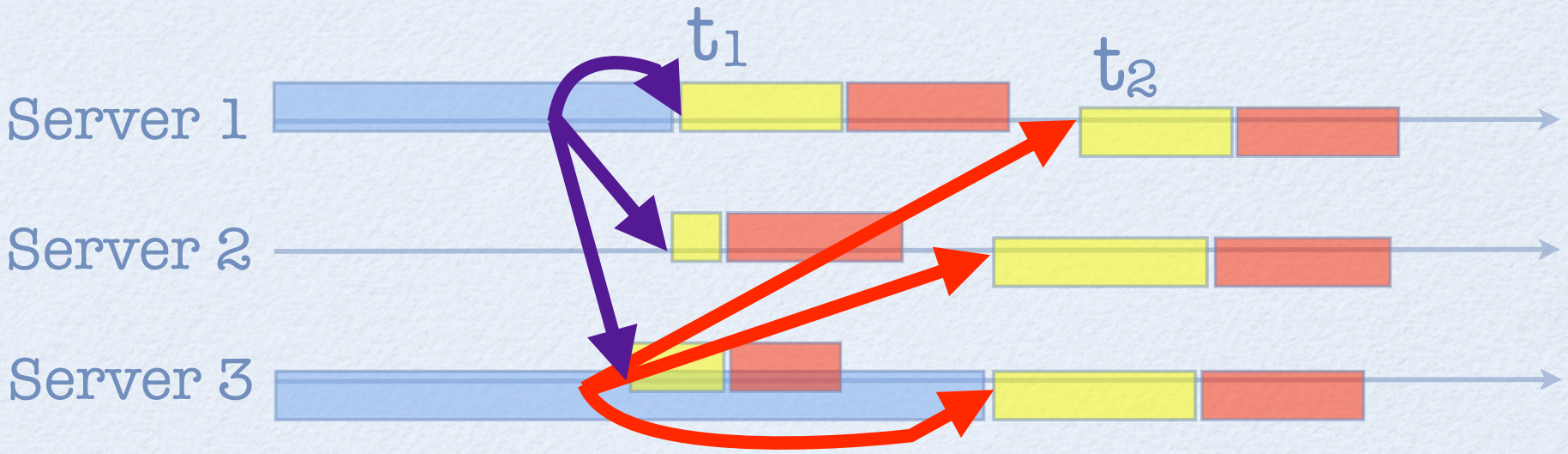
- Atomic broadcast-based certification
  - Example
    - Transaction t: read(x); write(y,-)
    - Transaction t': read(y); write(x,-)
    - t and t' are concurrent (i.e., neither t→t' nor t'→t)
    - All servers validate t and t' in the same order
    - Assume servers validate t and then t'
    - What transaction(s) commit/abort?

Deferred update replication

- Reordering-based certification
  - Let t and t' be two concurrent transactions
    - t executes read(x); write(y,-) and
    - t' executes read(y); write(z,-)
  - (a) t is delivered and certified before t'
    - t passes certification, but
    - t' does not: $RS(t') \cap WS(t) = \{ y \} \neq \emptyset$
  - (b) t' is delivered and certified before t
    - t' passes certification and
    - t passes certification(!): $RS(t) \cap WS(t') = \emptyset$

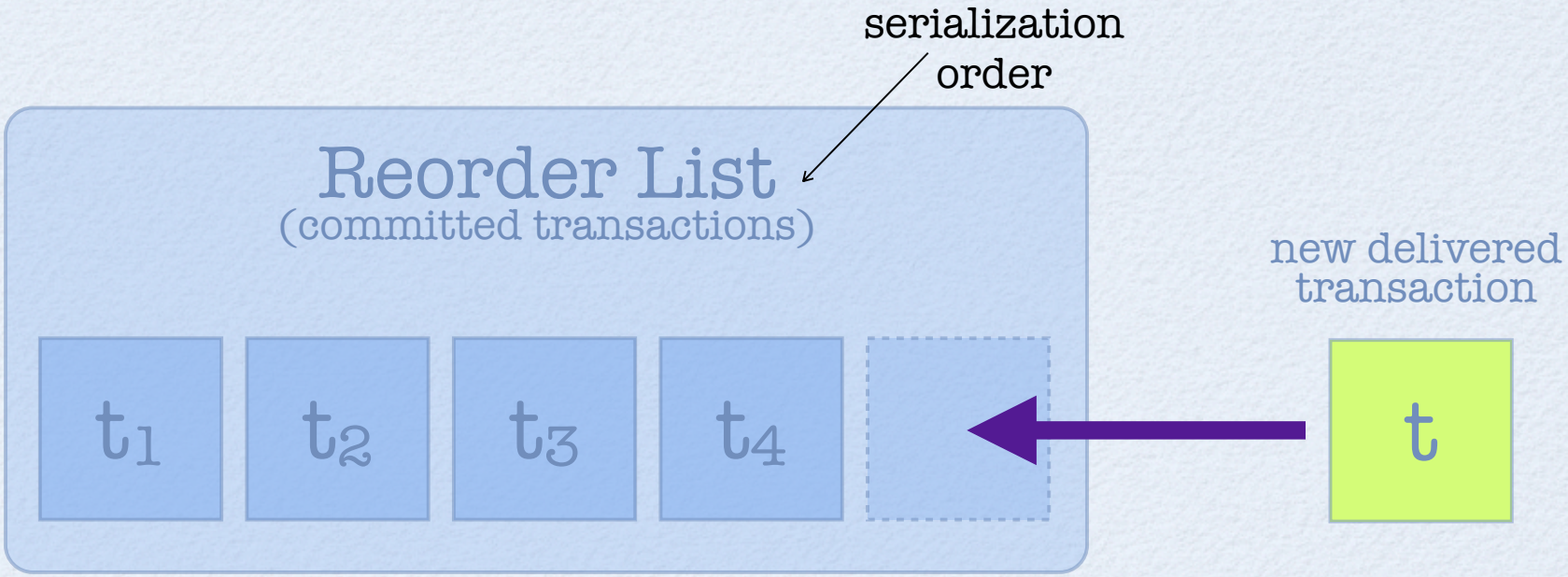
Deferred update replication

- Serialization order of transactions
  - Without reordering
    - Order given by abcast
    - Ex.: t_1 followed by t_2
  - With reordering
    - Either t_1 followed by t_2 or t_2 followed by t_1



Deferred update replication

- Reordering-based certification

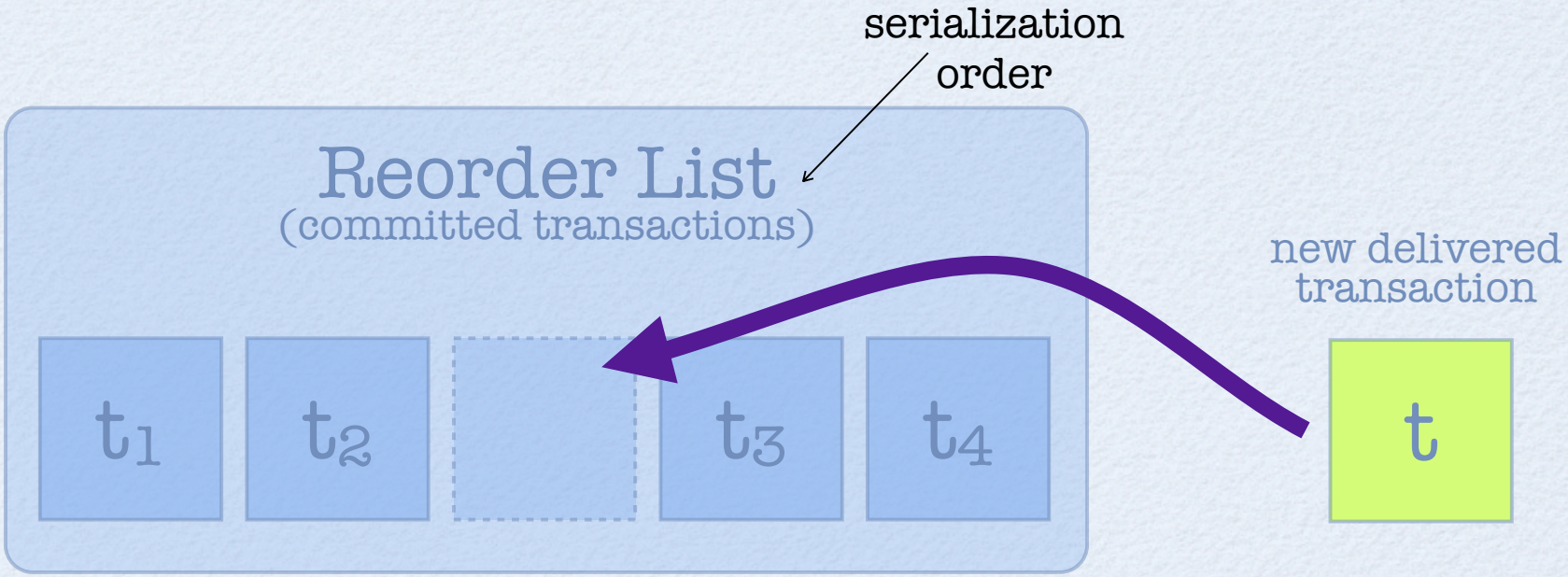


Condition for committing t (simplified):

$RS(t) \cap (WS(t_1) \cup WS(t_2) \cup WS(t_3) \cup WS(t_4)) = \emptyset$

Deferred update replication

- Reordering-based certification



Condition for committing t (simplified):

$RS(t) \cap (WS(t_1) \cup WS(t_2)) = \emptyset$ and

$WS(t) \cap (RS(t_3) \cup RS(t_4)) = \emptyset$

Deferred update replication

- Reordering-based certification

$$\forall t \forall s : Committing(t,s) \rightsquigarrow Committed(t,s) \equiv$$
$$\left[\begin{array}{l} \exists i,\ 0 \leq i < count,\ \text{s.t. } \forall t' \in RL_s : \\ \\ pos(t') < i \Rightarrow t' \rightarrow t \vee WS(t') \cap RS(t) = \emptyset \ \wedge \\ \qquad\qquad \wedge \\ pos(t') \geq i \Rightarrow \left(\begin{array}{c} (t' \not\rightarrow t \vee WS(t') \cap RS(t) = \emptyset) \\ \wedge \\ WS(t) \cap RS(t') = \emptyset \end{array}\right) \end{array}\right]$$

Deferred update replication

- Generalized reordering

Deferred update replication

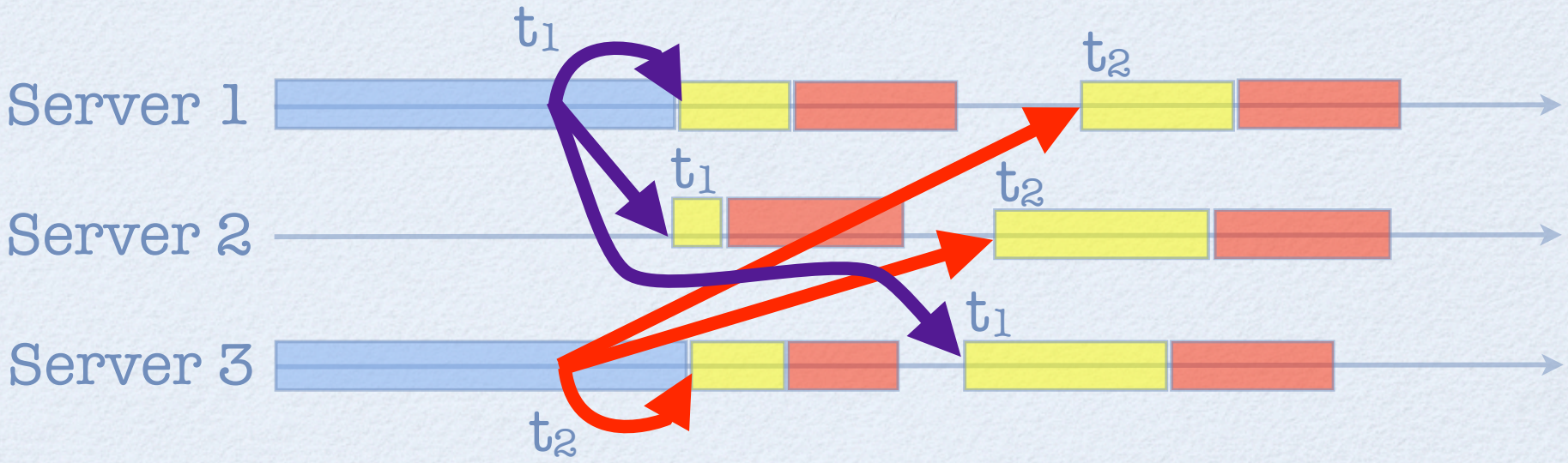
- Termination based on Generic Broadcast
  - Generic broadcast
    - Conflict relation ~ between messages
    - Properties
      - **Agreement**: Either all servers deliver m or no server delivers m
      - **Total order**: If messages m and m' conflict, then any two servers deliver them in the same order

Deferred update replication

- Generic broadcast
  - Motivation
    - Ordering messages is more expensive than not ordering messages
    - Messages should only be ordered when needed (as defined by the application)
  - Atomic broadcast is a special case of generic broadcast where all messages must be ordered

Deferred update replication

- Conflict relation between transactions
  - Assume transactions t_1 and t_2 don't conflict



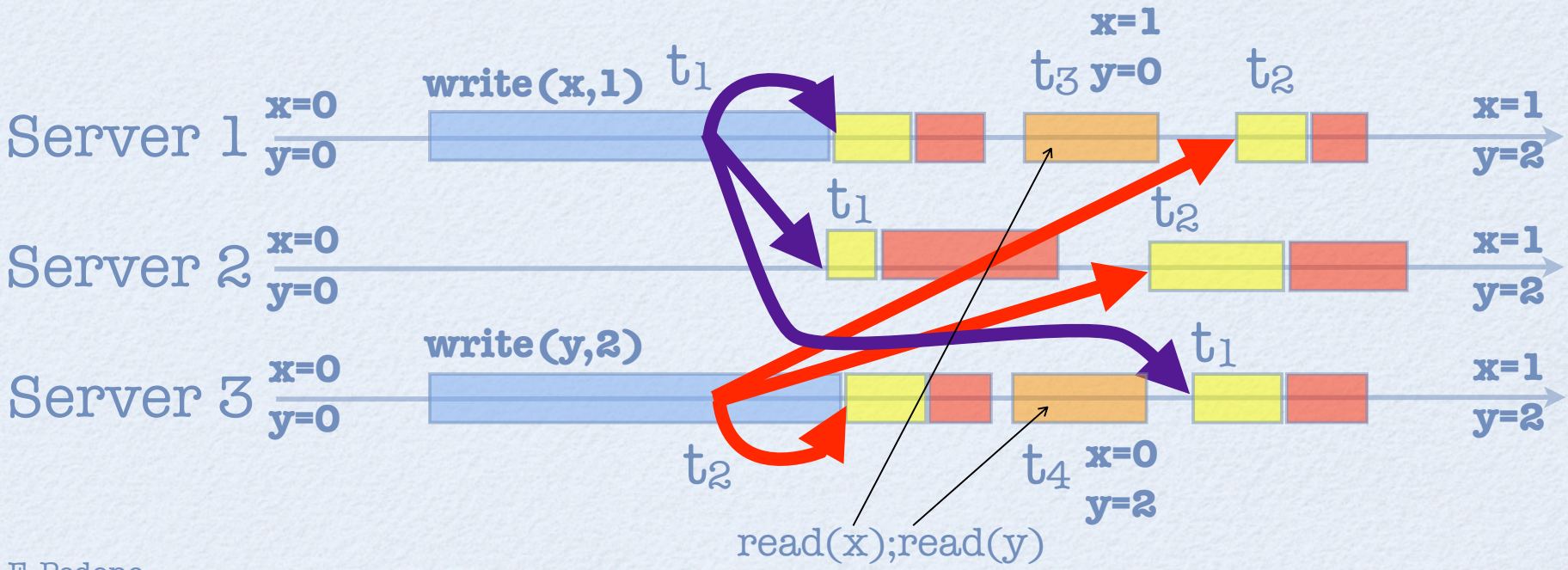
Deferred update replication

- Conflict relation between transactions
  - m:t means message m relays transaction t
  - if t conflicts with t' (i.e., m:t ~ m':t'), then they are delivered and certified in the same order

$$m:t \sim m':t' \equiv \left[\begin{array}{c} RS(t) \cap WS(t') \neq \emptyset \\ \vee \\ WS(t) \cap RS(t') \neq \emptyset \\ \vee \\ WS(t) \cap WS(t') \neq \emptyset \end{array}\right]$$

Deferred update replication

- Read-only transactions
  - Local execution without certification is not permitted
  - Different states of the database could be observed



Deferred update replication

- Read-only transactions
  - Optimistic solution
    - Broadcast and certify read-only transactions
  - Pessimistic solution
    - Pre-declare items to be read (readset)
    - Broadcast transaction before execution
    - Executed by one server only
    - Never aborted

Outline

- Motivation
- Replication model
- From objects to databases
- Deferred update replication
- **Final remarks**

Final remarks

- Object and database replication
  - Different goals
    - Fault tolerance × fault tolerance & performance
  - Different consistency models
    - Linearizability & sequential consistency × serializability
  - Different algorithms
    - Primary-backup & active replication × deferred update replication

Final remarks

- Consistency models
  - Relationship between L/SC and SR
- Replication algorithms
  - Unifying framework for object and database replication
  - Multi-primary passive replication

Final remarks

- Replication and group communication
  - A happy union
  - Group communication -- atomic broadcast -- leads to modular and efficient protocols (e.g., fewer aborts than atomic commit)
  - Replication has motivated more powerful and efficient group communication protocols (e.g., generic and optimistic primitives)

References

- Chapter 1:
  Consistency Models for Replicated Data
  A. Fekete and K. Ramamritham
- Chapter 2:
  Replication Techniques for Availability
  R. van Renesse and R. Guerraoui
- Chapter 11:
  From Object Replication to Database Replication
  F. Pedone and A. Schiper